Universidad
Carlos III de Madrid

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Diego Trompeta Urretavizcaya

Tutor: Jorge Ruiz Magaña

Conceptualización, análisis, diseño e implementación de un videojuego para Android

Agradecimientos

Es una tarea dificil, la de echar la vista atrás y buscar gente a quien se quiera dar las gracias. No son pocos los años que han pasado desde que comencé mi educación y mis agradecimientos no se limitan ni mucho menos a estos cuatro últimos años en la universidad. Sin duda habrá personas que se quedarán fuera de esta mención, pero no por ello su compañía y cariño han supuesto menos para mí que los de las que sí aparezcan.

En primer lugar, y como no podía ser de otra forma, quiero dar las gracias a mi abuelo, Jose María. No son pocos los años que llevo ya sin ti. Nunca olvidaré cuando me decías, que tenía que ser ingeniero y arquitecto. Pues bien, abuelo, ya estoy a la mitad, aunque la segunda parte quizá tarde un poco. Nunca te olvidaré. Muchísimas gracias por todo.

En segundo lugar, quiero agradecer a mis padres, Aquilino y Begoña, que hayan hecho posible que esté en este momento escribiendo estas palabras. De no ser por ellos de seguro me habría sido imposible. Gracias.

En tercer lugar, a mi tía Soraya. Una bellisima persona que siempre ha estado conmigo, en los buenos y en los malos momentos. Siempre a mi lado cuando he necesitado ayuda para lo que fuese. Siempre enseñándome cosas nuevas. Siempre guiándome por el buen camino. Muchas gracias, Soraya.

En cuarto lugar, a toda mi familia, a todos aquellos que en mayor o menor medida, me han ayudado y apoyado, especialente a mi hermana, Clara, a la cual quiero con locura. Muchas gracias a todos.

En quinto lugar, a todo lo que me ha aportado la Universidad Carlos III. A mis profesores, a mi tutor Jorge, por darme esta inmensa oportunidad y, en especial, a mis amigos Jorge, Pablo y Alberto, sin los cuales no habría podido vivir esta preciosa experiencia que es la universidad. Gracias, de verdad.

Y por último y para terminar, a esa persona especial, que hace mi vida más y más feliz cada día que pasa junto a mí. Muchísimas gracias por todos estos días juntos, Alejandra.

“No hay sangre en el suelo, ni ruido en las calles,
hemos cambiado de Columbia, otro tiempo, otras elecciones.
Una elección basta para abrir un sinfin de posibilidades,
y nosotros podemos explorarlas.”

Elisabeth
Bioshock Infinite

Resumen

Este proyecto consiste en la realización de un videojuego para dispositivos móviles, concretamente *Android*, de género *endless run*, que sea atractivo e intuitivo para el mayor número de público posible, asegurando de este modo la creación de una necesidad que poder explotar.

Para llevar a cabo este proyecto se necesitan las herramientas de desarrollo de *Android*, así como conocimientos sobre la programación en este sistema, que será necesario ampliar, y conocimientos sobre la programación de videojuegos, que será necesario adquirir.

Además, se realiza un estudio de mercado (análisis del estado del arte) sobre los videojuegos existentes con más renombre dentro del género *endless run*, con el fin de comprender las ventajas y desventajas que estos juegos presentan y cómo han afectado a su éxito dentro de la sociedad.

La frase que resume la intención de este trabajo es, por tanto: conceptualización, análisis, diseño e implementación de un juego tipo *endless run* para *Android*.

Palabras clave

Smartphone, videojuego, *endless run*, *Android*.

Índice general

Lista de figuras

Lista de tablas

Capítulo I: Introducción

El objetivo de este capítulo es realizar una breve introducción al tema tratado en el presente documento, exponer la motivación que impulsó la realización del trabajo, explicar el objetivo principal marcado, así como las metas personales planteadas, describir las fases de trabajo establecidas y la metodología seguida durante la elaboración de la aplicación. Para finalizar el capítulo, se enumerarán los medios empleados durante la realización del trabajo y se describirá brevemente la estructura del documento.

I.I Introducción

Los dispositivos móviles forman parte de nuestro día a día, de tal forma que tendemos a olvidarnos de que, no hace mucho tiempo, esto no era así. En 1983, con la venta del primer teléfono móvil, nadie se hubiese imaginado que, pocos años más tarde existirían más de esos dispositivos, que personas en el mundo. Por aquella época, los teléfonos móviles quedaban relegados a grandes ricos y empresarios, quienes requerían estar en constante contacto. Años más tarde, con las reducciones de precio y tamaño, estos dispositivos se extendieron rápidamente por todo el mundo.

Con esto se pretende describir el proceso de creación de una necesidad, el cual se tendrá muy en mente durante todo el desarrollo de este proyecto.

Por otro lado, el mercado convencional de los videojuegos está, lentamente, empezando a compartir público con un nuevo mercado enfocado a los dispositivos móviles.

Por regla general, las pequeñas desarrolladoras se han adueñado de este nuevo mercado, adaptándolo a una dinámica distinta a lo que venía siendo común en el mercado clásico. Juegos rápidos, sencillos y dinámicos, son los que triunfan en este mercado. Juegos realizados con presupuesto limitado y grandes restricciones estructurales, al contrario de lo que ocurre en el mercado clásico.

Con este proyecto se pretende:

- Crear un videojuego para dispositivos móviles, que aproveche las ventajas del mercado tradicional de los videojuegos, llevándolas a un nuevo terreno aún sin explotar.

I.II Motivación

Desde bien pequeño siempre tuve claro que iba a estudiar ingeniería informática. Nunca he albergado duda alguna sobre ello y, cuando llegó el momento, entré a la carrera sin pensármelo. No obstante, no ha sido hasta hace relativamente poco, en comparación al tiempo que hace desde que sé que iba a estudiar esta carrera, que he tenido claro que me quería dedicar al mundo del videojuego. Por este motivo, cuando fui a buscar un proyecto en el tablón, mi único interés era el de encontrar uno que me permitiese comenzar en el mundo del videojuego. Este mundo no es, para nada, fácil. Iniciarse en él, sin tener un mínimo de ayuda o guía, en el momento en que nos encontramos, con la industria tan ampliamente desarrollada, no es una tarea banal. Por este motivo, cuando encontré el proyecto ofrecido por mi tutor, Jorge Ruiz, vi una oportunidad perfecta. Haber podido contar con la ayuda y el apoyo de un profesional me ha aportado cierta tranquilidad, lo que ha hecho más llevadera la iniciación en este campo.

Por otra parte, mi fervorosa afición por los videojuegos me ha llevado a plantearme en bastantes ocasiones cómo poder hacerles un homenaje, y la realización de este Trabajo de Fin de Grado ha supuesto la oportunidad perfecta para ello.

I.III Objetivos principales y metas personales

Los objetivos principales de este proyecto son:

- Realizar un videojuego para dispositivos móviles *Android*, teniendo por objetivo un público con un margen de edad lo más amplio posible y con un resultado atractivo para todo ese público.
- Exportar los modelos exitosos del mercado tradicional del videojuego a un nuevo mercado centrado en los dispositivos móviles.
- Aprovechar un sector del mercado no explotado para crear e introducir una necesidad, que pueda ser aprovechada posteriormente comercializando nuestro producto.

Por último, las metas personales fijadas son las siguientes:

- Iniciarme en el mundo del desarrollo de videojuegos.
- Ampliar mis conocimientos de programación en Android hacia otro tipo de paradigmas de desarrollo de aplicaciones muy diferentes a las estudiadas hasta el momento.
- Conseguir un grado aceptable de conocimientos en la programación de videojuegos para permitirme la entrada profesional en este campo.
- Conseguir realizar un homenaje al mundo del videojuego.

I.IV Fases de trabajo

El proceso de desarrollo se ha dividido en cuatro fases bien diferenciadas que se procede a describir a continuación:

Fase 1: Conceptualización

En esta fase se estableció el sistema para el cual se pretendía desarrollar el videojuego. Tras una reunión inicial se estableció que éste sería *Android*.

Posteriormente, se comenzó a sopesar el género del juego a desarrollar. Por ser el primer videojuego que se iba a realizar, se optó por hacerlo del género *endless run*[1]dado que las características del mismo permitían enfocarlo a un gran público, con un mayor margen de edad. Además el hecho de no tener que depender de un hilo de historia como en otro tipo de juegos permitía centrarse más en ofrecer un mayor número de posibilidades.

Fase 2: Análisis y diseño

Durante esta fase se comenzó a establecer cómo iba a ser el juego. Para ello fue necesario tomar ciertas decisiones, descritas en capítulos posteriores. Asimismo, se crearon bocetos y prototipos de la aplicación, y se buscaron los elementos que posteriormente se utilizarían en la fase de implementación, así como se crearon los diferentes gráficos que se utilizaron posteriormente. Por último, se trabajó en desarrollar el bucle del videojuego, las rutinas gráficas y los sonidos y sprites.

Fase 3: Implementación y pruebas

El proceso de implementación pasó por tres etapas bien definidas: creación del entorno, creación del juego y unión de ambos. Durante las dos primeras fases se llevaron a cabo las pruebas unitarias, las de integración y parte de las pruebas de sistema. Durante la última etapa se realizaron el resto de las pruebas de sistema.

Fase 4: Redacción de la memoria

Una vez realizada y probada la aplicación se presentó el resultado al tutor, quien dio el visto bueno para comenzar a escribir la presente memoria.

[1] Género caracterizado por la dinámica "correr hasta fallar", es decir, el personaje protagonista avanza por un mundo esquivando enemigos y, normalmente, recolectando algún ítem, hasta que, simplemente, muere.

I.V Metodología

Dada la naturaleza del proyecto, en el cual no se establecieron requisitos iniciales, se decidió adoptar una metodología desarrollada de forma personal adaptada a partir de la metodología de desarrollo ágil *Extreme Programming*[1], la cual se basa en realizar una programación lo más simple posible para alcanzar el objetivo requerido [2]. Una de las bases de esta metodología (como de tantas otras metodologías de desarrollo ágil) consiste en considerar al cliente como uno más dentro del equipo de desarrollo. De este modo, se contará con un equipo de usuarios en representación a los clientes (posibles compradores del producto), los cuales colaborarán durante la fase de pruebas del producto.

I.VI Medios empleados

A continuación se enumeran los medios utilizados durante la elaboración de este proyecto:

Hardware

- Ordenador de sobremesa *Intel* i7-3770
- Ordenador portátil Compaq Pavilion g6t-2000, Intel i3-2350M
- Teléfono móvil *Samsung Galaxy* S2 GT-I9100
- Teléfono móvil *Huawei Ascend* G300
- Teléfono móvil *Samsung Galaxy Ace* S5830

Software

- Sistema Operativo *Windows* 7 *Professional* 64 bits x2
- Sistema Operativo *Android* 4.1.2 *Jelly Bean*
- Sistema Operativo *Android* 4.0.3 *Ice Cream Sandwich*
- Sistema Operativo *Android* 2.3.3 *Gingerbread*
- *Adobe Photoshop* CS4 (diseño de fondos y botones)
- *Paint*.NET v3.5.10 (diseño de personajes)

I.VII Estructura del documento

A continuación, se realizará una descripción de los siguientes capítulos de esta memoria:

- ***Capítulo 2: Estado del arte:*** en este capítulo se analizarán el entorno de desarrollo seleccionado y aplicaciones similares a la aplicación a desarrollar, y se realizará una valoración de lo que las metodologías ágiles han aportado al desarrollo del proyecto.
- ***Capítulo 3: Especificación de requisitos:*** este capítulo incluirá todo lo relacionado con la extracción de requisitos para la aplicación, la especificación de los mismos y las matrices que los relacionan.
- ***Capítulo 4: Diseño de la aplicación:*** en este capítulo se presentan los prototipos de la aplicación, las decisiones de diseño tomadas, los diagramas de clases y los diagramas de secuencia de navegación de la aplicación.
- ***Capítulo 5: Pruebas de la aplicación:*** en este capítulo se exponen los resultados de las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación.
- ***Capítulo 6: Conclusiones y líneas futuras:*** en este capítulo se exponen las conclusiones extraídas tras la elaboración del trabajo y las líneas futuras de desarrollo posibles para la aplicación.

Capítulo II: Estado del arte

El objetivo de este capítulo es analizar los entornos de desarrollo móvil disponibles y exponer el elegido, realizar un análisis de aplicaciones similares a la que se pretende desarrollar y describir cómo las metodologías ágiles han colaborado en el desarrollo del proyecto. Para finalizar, se expondrán brevemente las conclusiones extraídas de este análisis.

II.I Entorno de desarrollo móvil seleccionado

En este apartado se analizarán los diferentes entornos móviles en su estado actual, se expondrá una breve introducción a *Android* y, por último, se describirá el entorno de desarrollo seleccionado y las razones por las que se ha elegido ese entorno y no otro de entre los existentes.

II.I.I Entornos móviles

Para realizar el análisis sobre los entornos móviles existentes partiremos desde el punto de vista de terminales actuales en el mercado. De este modo, a fecha de mayo de 2013, los terminales más relevantes en el mercado han sido, por orden decreciente, *iOS*, *Android*, *Java ME* y *Symbian*.

Tal y como se muestra en el siguiente gráfico, *iOS* ocupa un amplio 59% del mercado, mientras que *Android* se queda en un 24%:

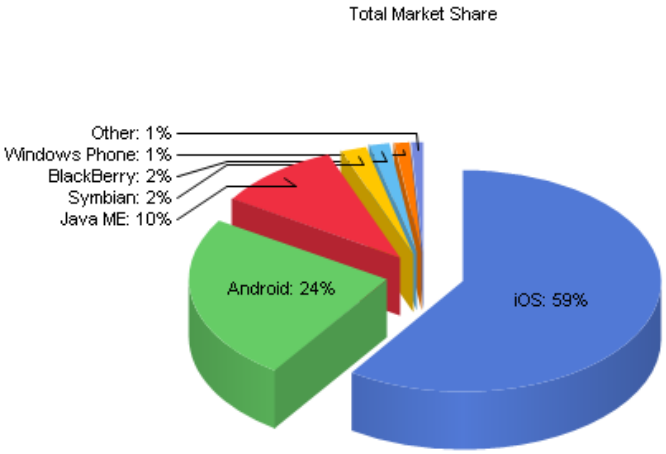


Figura II.I.I.I Terminales en uso Mayo 2013[2]

No sorprende encontrar a estos dos gigantes en los dos primeros puestos, pero sí el dato de que *Java ME* se sitúe como tercer sistema operativo más utilizado. Eso es un dato erróneo, ya que *Java ME* no es un sistema operativo como tal, sino una capa middleware que viene instalada por defecto en varios sistemas operativos móviles, tales como *Symbian*, *Windows Phone* y *Blackberry*.

[2] http://www.netmarketshare.com/

Por otro lado, si prestamos atención a la cuota de dispositivos vendidos, sobre todo en España, podemos observar que *Android*, a fecha de marzo de 2013, posee un 93.5% del total de ventas de dispositivos móviles.

Spain	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	5.4	3.2	- 2.2
Android	76.8	93.5	16.7
RIM	7.8	0.2	- 7.6
Symbian	8.1	1.5	- 6.6
Windows	1.8	1.3	- 0.5
Other	0.0	0.3	0.3

EU5	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	20.4	19.4	- 1.0
Android	58.1	68.8	10.7
RIM	8.1	2.7	- 5.4
Symbian	7.2	1.6	- 5.6
Windows	4.1	6.5	2.5
Other	2.2	1.1	- 1.2

Tabla II.I.I.I Ventas de terminales marzo 2012-2013[3]

Por lo tanto, valorando en conjunto el número de terminales existentes y las ventas actuales en España, se ha decidido tener por objetivo *Android* para la realización de la aplicación. Además, el previo conocimiento de *Java* y el disponer de un dispositivo *Android* fomentaron esta opción por encima de las demás.

[3] http://www.xatakandroid.com/mercado/android-no-toca-techo-en-espana-ya-vende-el-93-5-de-los-smartphones

II.I.II Visión general de *Android*

Como ya se ha explicado en la sección anterior, el sistema operativo escogido para el desarrollo de la aplicación ha sido *Android*, por su gran número de ventas durante este año 2013 y su aceptable cuota de mercado global.

Android es un sistema operativo móvil basado en Linux y desarrollado principalmente para dispositivos móviles con pantalla táctil como smartphones y tablets. Fue presentado oficialmente en 2007 y en octubre de 2008 se vende el primer terminal con sistema operativo *Android*. La fundación *Open Handset Alliance*, liderada por Google, es la encargada de desarrollar *Android*.

Las diferentes versiones de *Android* reciben nombres de postres en inglés. De este modo, cada nueva versión obtiene un nombre compuesto por su código de versión y un postre inglés que empieza por una letra distinta siguiendo orden alfabético. Así mismo, aproximadamente a cada nueva versión le corresponde un nuevo nivel dentro de sus APIs de programación.

En la siguiente tabla se muestran todas las versiones de *Android* hasta la fecha:

Nombre	Código	Nivel de API
Apple Pie	1.0	1
Banana Bread	1.1	2
Cupcake	1.5	3
Donut	1.6	4
Eclair	2.0 – 2.1	5, 6, 7
Froyo	2.2 – 2.2.3	8
Gingerbread	2.3 – 2.3.7	9, 10
Honeycomb	3.0 – 3.2.6	11, 12, 13
Ice Cream Sandwich	4.0 – 4.0.4	14, 15
Jelly Bean	4.1 - xx	16, 17

Tabla II.I.II.I Versiones y APIs de Android

La versión más actual de *Android* es, como muestra la tabla anterior, *Jelly Bean*, que es una versión mejorada de *Ice Cream Sandwich*, donde se introdujeron, por

primera vez en los smartphones, las mejoras provenientes de la línea de desarrollo de *Android* para tablets (versiones 3.x) y fue el punto de unificación de ambas vías de desarrollo.

Dado que *Android* no es un sistema operativo exclusivo de un único fabricante de terminales y ya que nuevos terminales salen continuamente al mercado, existe una gran fragmentación de versiones dentro del sector de *Android*. Esto se ve reflejado a la hora de planificar el desarrollo de aplicaciones para estos terminales, ya que las diferentes versiones de las APIs disponibles provocan que las funciones disponibles cambien de una versión a otra. En la siguiente tabla se muestra la fragmentación del mercado de *Android* en sus diferentes versiones a fecha de mayo de 2013:

Nombre	Nivel de API	Cuota de mercado (en %)
Donut	4	0.1
Eclair	5, 6, 7	1.7
Froyo	8	3.7
Gingerbread	9, 10	38.5
Honeycomb	11, 12, 13	0.1
Ice Cream Sandwich	14, 15	27.5
Jelly Bean	16, 17	28.4

Tabla II.I.II.II Cuota de mercado de las versiones de *Android* mayo 2013[4]

Para la elaboración de la aplicación a desarrollar se ha escogido como versión objetivo 4.2.2 *Jelly Bean*, por ser la versión más actualizada de *Android*. No obstante, para cumplir nuestro objetivo de llegar al más amplio rango de público se tendrán en cuenta las restricciones de las diferentes versiones de *Android*, adaptando la aplicación a cada versión en el caso de que fuese necesario.

[4] http://es.wikipedia.org/wiki/Android

II.I.III Estructura de *Android*

El sistema operativo *Android* está formado por varios componentes que se muestran en la siguiente figura y se explican posteriormente:

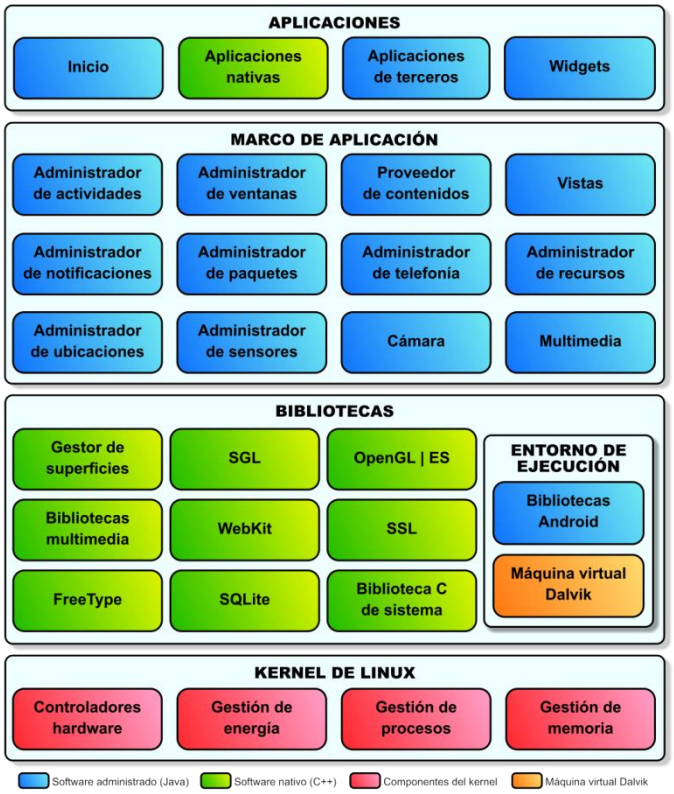


Figura II.I.III.I Estructura de *Android* [3]

Aplicaciones

Conjunto de aplicaciones instaladas por defecto en *Android*, que facilitan al usuario determinadas tareas que ofrece por defecto el sistema operativo, como pueden ser: llamar por teléfono, consultar correo, administrar contactos, etc.

Estas aplicaciones están descritas en *Java* y *C++*.

Marco de Aplicación

El marco de aplicación consta de un conjunto de APIs a disposición de los programadores y desarrolladores que facilitan el acceso a funciones más complejas del dispositivo, las cuales son utilizadas de forma común por la mayoría de las aplicaciones. Estas APIs facilitan el acceso a funciones tales como servicios multimedia, gestión de actividades, gestión de notificaciones, etc.

Estas APIs están descritas en *Java*.

Bibliotecas

Bibliotecas a disposición del desarrollador utilizadas por algunos componentes del sistema operativo.

Están descritas en el lenguaje nativo del procesador, en este caso, en *C/C++*.

Entorno de ejecución

Esta capa está formada por dos componentes, las bibliotecas de *Android* y la máquina virtual *Dalvik*.

Dada la diversidad de dispositivos donde se iba a ejecutar *Android* y debido a las limitaciones de estos, no fue posible utilizar la máquina virtual estándar de *Java*. Por este motivo, *Google* decidió crear la máquina virtual *Dalvik*, una versión de la original de *Java*.

Cada aplicación en *Android* se ejecuta sobre una máquina virtual *Dalvik* distinta. El sistema operativo es el que se encarga de gestionar cada una de estas máquinas, aislándolas entre sí. Este proceso está basado en la utilización de una pequeña región de memoria para cada aplicación en ejecución denominada *sandboxing*, que provee al sistema operativo de seguridad y versatilidad.

Kernel de *Linux*

El núcleo de *Android* está formado por el kernel de *Linux* en la versión 2.6, el cual provee a *Android* de servicios tales como la seguridad, manejo de la memoria y procesos, la pila de protocolos de red y el soporte de drivers para dispositivos. Esta capa actúa también como capa de abstracción entre el hardware y el software.

Esta estructura proporciona flexibilidad frente a la implementación de nuevas aplicaciones, así como videojuegos, tanto por parte de desarrolladores oficiales como de terceros, dado que el hardware sobre el que se implementa el sistema *Android* como el modelo de programación lo soportan.

II.I.IV Entorno de desarrollo móvil integrado

El entorno de desarrollo escogido para la realización de este trabajo ha sido *Eclipse Juno* ya que ha sido la herramienta utilizada de forma habitual durante toda la carrera y ya se estaba familiarizado con ella.

Para la instalación y actualización de la última versión disponible y para evitar los posibles problemas a la hora de instalar el SDK de *Android*, se optó por descargar la versión ya preparada que proporciona *Google* en su página de desarrollo de *Android*. Una vez se hubo descargado la versión correcta (64 bits), se procedió a actualizar el entorno de desarrollo a la última versión disponible.

Por defecto, el SDK de *Android* sólo incluía el API 15, por lo que fue necesario descargar e instalar los APIs 17 y 10. Se decidió descargar el API 10 por ser la última versión de smartphones previa a la unificación entre tablets y smartphones y la que más cuota de mercado posee actualmente. De este modo, a la hora de realizar las pruebas se podría ejecutar sobre una máquina virtual con versión *Gingerbread* para comprobar el correcto funcionamiento de la aplicación y garantizar que se cumple nuestro objetivo principal.

Para finalizar, cabe destacar la utilidad que ha suministrado la herramienta *LogCat*, la cual imprime trazas de las aplicaciones en tiempo de ejecución. Gracias a ella, ha sido posible realizar gran parte de las pruebas unitarias y de integración en tiempo de desarrollo, así como el resto de las pruebas en fase de validación.

II.II Análisis de aplicaciones en base al objetivo establecido

Una vez que ya conocemos el entorno de desarrollo objetivo nos encontramos en situación de explorar el mercado, para realizar un análisis sobre las aplicaciones existentes similares a la que se pretende desarrollar.

Recordando lo expuesto en el capítulo anterior, se pretende realizar un videojuego del género *endless running* que sirva como homenaje a los grandes videojuegos y que sea atractivo para el mayor rango de público posible. Además se busca unificar la vía tradicional de desarrollo de videojuegos con los dispositivos móviles de tal forma que se cree una nueva necesidad que pueda ser explotada posteriormente.

En pos de conseguir estos objetivos se ha procedido a realizar un análisis buscando tanto en el buscador web *Google*, como en la tienda de *Android*, *Google Play*, por los siguientes tags:

- *endless running*
- *tribute game*
- *endless tribute*
- *nintendo*
- *kirby*
- *mario*
- *pikachu*
- *pokémon*

Hemos recurrido a revistas electrónicas de videojuegos para obtener una opinión experta sobre los juegos más emblemáticos de este género.

Cabe destacar, previamente a la exposición de los resultados obtenidos, que no se ha encontrado ninguna referencia a juegos de género *endless running* y prácticamente de ningún otro tipo para los tags *nintendo*, *kirby*, *mario*, *pikachu* y *pokémon*, lo que establece un terreno firme que nos indica que este mercado aún no ha sido explotado y que nuestra aplicación podría tener éxito.

A continuación se exponen los diferentes juegos analizados como resultado de este estudio. Se prestará especial atención a lo intuitivo de cada juego, así como a la dificultad del mismo, ya que nuestro interés es el de alcanzar al mayor público posible.

II.II.I *The Impossible Game*

En su origen, *The Impossible Game* [4] fue desarrollado para *Xbox 360* como juego *indie*. Tras su lanzamiento y rápido éxito fue portado a los principales mercados móviles: *iOS*, *Android* y *Windows Phone*.

Este juego parte del concepto básico de mezclar la música con el juego, lo cual no debe confundir ni hacer creer que es un juego de tocar música. Según se va avanzando en los niveles, esta irá acompañando al jugador y cambiando según la acción en el nivel se vuelva más o menos frenética.

Se trata de un juego al más puro estilo plataformas. En él somos un cubo naranja que avanza sin parar y nuestra única acción posible es la de tocar la pantalla para realizar un salto o mantenerla pulsada para encadenar saltos continuados. Durante el avance de sus cinco niveles nos iremos encontrando con triángulos y cuadrados. Si entramos en contacto con los triángulos, nuestro cuadrado naranja estallará en mil pedazos haciendo que tengamos que volver a empezar desde el inicio. Por el contrario, los cuadrados nos ofrecen la posibilidad de subirnos encima de ellos para intentar esquivar zonas de peligro, señaladas por un suelo negro, en contraste al blanco en el cual estaremos seguros. Cabe destacar el color de los "enemigos", ya que al ser de tonalidades grises-negras favorece que el reconocimiento de lo que es malo dentro del juego resulte plenamente intuitivo.

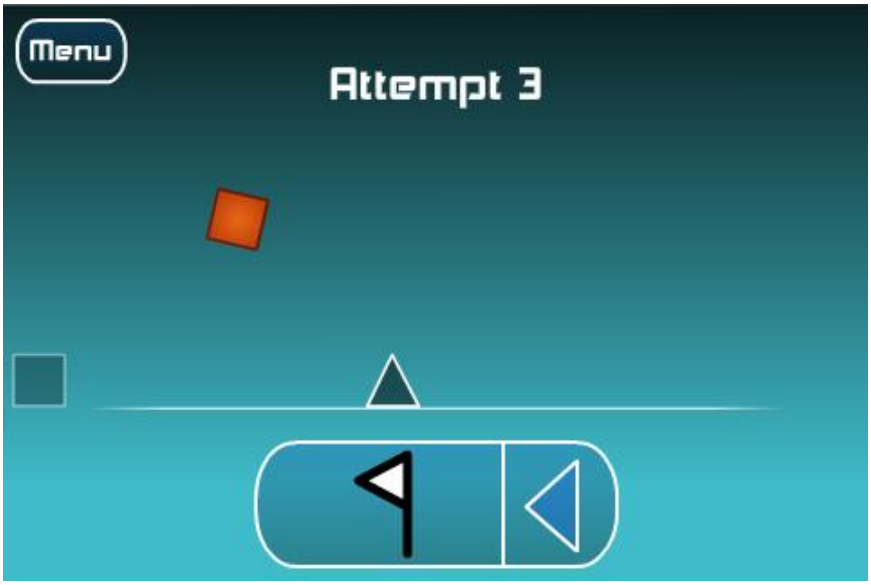


Figura II.II.I.I *The Impossible Game*

No obstante, es necesario destacar que el nombre del juego hace honor a su dificultad a la perfección. Este puede llegar a ser en más de una ocasión frustrante, pues como ya hemos dicho si se muere por cualquier motivo es obligatorio volver a empezar el nivel desde el principio. Además, por si no fuese suficiente, los niveles están diseñados de tal forma que siempre hay saltos justos, numerosos obstáculos, caminos erróneos, y todo ello siempre acompañado por la frenética música, que en ocasiones ayuda al jugador a saber cuándo tiene que realizar un salto, pero en muchas otras no hace más que aumentar la tensión del jugador. Aun así, es necesario remarcar que la música es prácticamente el eje central del juego, ya que, aunque con ella pueda resultar frustrante jugar, sin ella se hace completamente imposible, ya que el oído es un punto muy fuerte que nos ayuda a anticiparnos a los obstáculos.

El juego cuenta con un modo de prácticas, accesible desde los propios niveles de juego, donde se nos permite ir colocando "checkpoints" en forma de banderas a lo largo de nuestro avance, que se encargarán de que si morimos no tengamos que repetir el nivel desde el principio, sino que renaceremos en el punto en que hayamos puesto el último banderín.

Figura II.II.I.II Modo de práctica de *The Impossible Game*

Con respecto al aparado gráfico del juego, cabe destacar la sencillez del mismo. Un fondo estático utilizando la técnica del gradiente de azul a negro nos acompaña durante todos los niveles dejando paso a un fondo exclusivamente negro en los menús

del juego. Con respecto a los propios menús hay que destacar de nuevo lo intuitivo de ellos, constando de cuatro opciones iniciales que nos permiten iniciar el juego, ver una rápida guía de cómo jugar, comprobar las medallas que hemos obtenido y ver unas breves estadísticas sobre el juego.

Como parte negativa, cabe destacar que la forma de acceder a los niveles de prácticas resulta en ciertas ocasiones demasiado molesta, ya que al estar situada en la parte inferior central de la pantalla durante los niveles provoca que, a veces, se pulse sin querer en esta zona entrando automáticamente en el modo de prácticas.

El juego se encuentra únicamente en inglés, aunque el nivel requerido para comprenderlo todo es muy básico. Se requiere *Android* 1.6 *Donut*, como mínimo, para ejecutar la aplicación.

II.II.II Temple Run y Temple Run 2

Dentro del prolífico género *endless run* en dispositivos móviles, gracias a lo fácilmente que se ajusta su mecánica a las características de las pantallas táctiles, encontramos dos grupos bien diferenciados, los juegos en 2D y los juegos en 3D. En el caso anterior de *The Impossible Game* nos encontramos ante un juego del tipo 2D. Sin embargo para la saga *Temple Run* nos encontramos ante un juego en 3D.

Temple Run [5] fue lanzado inicialmente en *iOS*, y posteriormente en *Android* cosechando un gran éxito de descargas en formato *free-to-play*. Al cabo de un tiempo y como continuación a la primera parte, se lanzó *Temple Run 2*, que no fue otra cosa más que una continuación "más y mejor", es decir, más posibilidades y mejor apartado técnico. Esta segunda entrega incluyó la posibilidad de realizar micro pagos para conseguir beneficios, también obtenibles de forma normal durante el juego, pero sin necesidad de luchar por ellos. Esta política de micro pagos ha hecho obtener unos beneficios considerables a *Imangi Studios*, responsables de ambas entregas.

Figura II.II.II.I Pantalla de título *Temple Run* y *Temple Run 2*

La dinámica de estos dos títulos consecutivos es algo más compleja que la expuesta en el caso anterior. En esta saga somos un arqueólogo que ha viajado a lo más profundo de una selva en busca de templos que explorar y conseguir sus tesoros. Una

vez el protagonista ha encontrado lo que fue a buscar, comienza a ser frenéticamente perseguido por una manada de "monos" protectores del templo, en la primera entrega de la saga, o por un "mono gigante", en la segunda. La participación del jugador empieza exactamente al inicio de la persecución y su objetivo es aguantar lo máximo posible escapando de sus persecutores a la par que tiene que esquivar obstáculos y recolectar *tokens*, que posteriormente podrá usar para obtener mejoras en el juego. Las acciones posibles a realizar por el jugador son algo más extensas que en *The Impossible Game*. En este caso, tendremos que inclinar el dispositivo a derecha o izquierda para recoger los *tokens* y deslizar el dedo por la pantalla táctil en cada una de las cuatro direcciones para realizar diferentes acciones. Si se desliza el dedo hacia arriba, el protagonista saltará un obstáculo que aparece por el suelo. Si, por el contrario, se desliza hacia abajo, el protagonista se deslizará por el suelo para esquivar un obstáculo "aéreo". En el caso de deslizar el dedo hacia la derecha o hacia la izquierda el personaje realizará un giro de 90 grados en la dirección deseada, permitiéndonos evitar un tramo de camino que nos guiaría hacia un precipicio.

Figura II.II.II.II *Temple Run* gráficos

En la segunda entrega de esta saga se incluyen ciertas novedades con respecto a la primera. El juego deja de ser exclusivamente a pie para alternar fases a bordo de una vagoneta al más puro estilo de mina antigua y fases de "tirolina" en las cuales el suelo desaparece forzando al protagonista a engancharse en una cuerda. Además se incluyen varios corredores desbloqueables así como un nuevo tipo de moneda obtenible de forma relativamente compleja en el juego normal, pero que podía conseguirse mediante micro pagos en la tienda del juego.

Figura II.II.II.III *Temple Run 2* gráficos

Con respecto al apartado gráfico, mientras que la primera entrega no destacó mucho, en la segunda sí que se prestó más atención a este aspecto, logrando resultados mucho más pulidos que su predecesora, así como entornos mucho más agradables a la vista, con cielos abiertos y escenarios mucho más variados. En cuanto a menús se refiere, cabe destacar la relativa complejidad de los mismos. Una persona adaptada al juego puede fácilmente navegar por ellos, pero para un usuario principiante pueden llegar a resultar complejos, lo que hace poco intuitiva la navegación por los mismos.

Si algo negativo se puede decir sobre esta segunda entrega, es sobre los ya explicados micro pagos, los cuales permiten conseguir altas puntuaciones (objetivo

único del juego) sin necesidad de esfuerzo alguno, lo cual ha sido ampliamente criticado por el sector de fans más purista.

Los textos del juego se encuentran escritos en español. Se necesita *Android* 2.1 *Eclair*, como mínimo, para poder ejecutar la aplicación.

II.II.III Jetpack Joyride

Una vez más nos encontramos ante un juego de género *endless run* en 2D. Éste es, a grandes rasgos, el más claro ejemplo de videojuego *endless run* puro, de todos los que se van a analizar.

Lanzado por *Halfbrick Studios* tras el relativo fracaso de *Monster Dash*, *Jetpack Joyride* [6] nos pone al control de *Barry Steakfries*, un comercial que se encuentra con problemas en su negocio. Por casualidad, Barry acude a un laboratorio secreto para intentar vender sus productos y allí descubre un experimento secreto, un jetpack "ametralladora". *Barry* decide robar este dispositivo y emprende la huida del laboratorio utilizándolo.

Figura II.II.III.I Título de *Jetpack Joyride*

La mecánica del juego es relativamente fácil e intuitiva. *Barry* correrá sin parar de izquierda a derecha, mientras que la labor del jugador consiste en esquivar las diferentes medidas de seguridad que intentan impedir la huida del protagonista. Con un toque en la pantalla, *Barry* dará un pequeño salto, mientras que si dejamos pulsada la pantalla el jetpack entrará en funcionamiento haciendo que *Barry* se eleve en el aire.

A lo largo de la partida nos iremos encontrando con diferentes elementos, como pueden ser monedas que deberemos recolectar para conseguir beneficios posteriores, tokens que podremos recoger para participar en una lotería al acabar el nivel con efectos potenciadores, científicos que inicialmente no supondrán ningún peligro y enemigos en forma de zonas electrificadas, misiles y láseres.

Además es posible encontrar a lo largo de la partida una serie de vehículos que alteran las mecánicas del juego y que van desde un pájaro mecánico, el cual se controla con toques consecutivos en la pantalla, hasta un traje gravitatorio que permite cambiar la dirección de la gravedad con un toque en la pantalla.

El objetivo del juego es, al igual que en los juegos *endless run* puros, aguantar el máximo tiempo posible sin morir recolectando el mayor número de ítems posibles.

Como extra, el juego propone la posibilidad de ir completando una serie de misiones, que se actualizan conforme se van completando, que pueden ir desde recolectar cierto número de tokens hasta esquivar por muy poco un número determinado de misiles y demás locuras.

Figura II.II.III.II Pantalla de juego de *Jetpack Joyride*

Una vez terminada la partida, se ofrece la posibilidad de adquirir ciertas mejoras utilizables por el personaje. Algunas de ellas no son más que simples complementos estéticos mientras que otras son mejoras efectivas que repercuten sobre el propio juego.

De nuevo, al igual que en el caso del *Temple Run 2*, se incluye la posibilidad de realizar micro pagos para conseguir oro del juego, con el fin de obtener mejoras. Este hecho no repercute gravemente en las posibilidades de cada jugador, ya que según se avanza en la partida la velocidad de esta va aumentando e independientemente de qué mejoras tengas, si no eres habilidoso no llegarás muy lejos.

En cuanto a los menús, hay que destacar que las muchas posibilidades que ofrece este juego se ven acompañadas por unos menús bastante extensos, lo que, a

priori, no resulta muy intuitivo si no tienes una experiencia previa en el mundo de los videojuegos.

Pese a todo ello, la mecánica de las misiones aporta un cierto grado de interés que incita a no querer dejar de jugar con el fin de completar, en cada partida, al menos una misión nueva. Este hecho fomenta que el usuario no se aburra rápidamente, por lo que se tendrá en cuenta de cara al proyecto.

Los textos del juego se encuentran escritos en español. Se necesita *Android* 2.2 *Froyo*, como mínimo, para poder ejecutar la aplicación.

II.II.IV Rayman Jungle Run

En contraposición a los juegos analizados hasta ahora, los cuales provienen todos de estudios independientes, este es el primero y único del género que se ha encontrado propiedad de un gran desarrollador, *Ubisoft*.

Hasta la fecha no son muchas las grandes compañías de videojuegos que se han planteado la posibilidad de saltar al mercado móvil y muchas menos las que ya lo han hecho. *Ubisoft* se propuso conseguir, con esta entrega, lo mismo que este trabajo pretende, migrar una vía de desarrollo exitosa del mercado del videojuego tradicional al nuevo modelo de mercado, es decir, aprovechar la necesidad existente de disponer de un dispositivo móvil para crear otra necesidad, la de disponer de su personaje estrella, *Rayman*[5], también en estos dispositivos.

Figura II.II.IV.I *Rayman Jungle Run* título

El lanzamiento de esta entrega coincidió con el lanzamiento para consolas de sobremesa del videojuego *Rayman Origins*. Ambas entregas comparten motor gráfico (*UbiArt Framework*) por lo que da la sensación de que el videojuego móvil sea igual de potente gráficamente que el de consolas. Esto no es del todo correcto, ya que en la versión de dispositivos móviles se redujo en gran medida el detalle de los entornos, para adaptar y optimizar la aplicación acorde a los limitados recursos de este tipo de dispositivos.

[5] http://es.wikipedia.org/wiki/Rayman

En cuanto a su género, *Rayman Jungle Run* se sitúa a medio camino entre plataformas y *endless run*, decantándose más hacia las plataformas, dado el factor limitado de sus niveles, ya que todos tienen inicio y fin, y duran aproximadamente un minuto.

La mecánica del juego es relativamente intuitiva. En él tomamos el papel de *Rayman*, quien correrá automáticamente por todo el nivel, normalmente de izquierda a derecha, pero en alguna excepción se dará la vuelta hacia el otro sentido. La labor del jugador consiste en dar toques en la pantalla para que *Rayman* salte esquivando los obstáculos o rebotando por las paredes. A medida que se va avanzando por los diferentes mundos del juego, *Rayman* va ganando capacidades, como puede ser planear o su clásico lanzamiento de puño. Esto hace que los niveles vayan incrementando su dificultad a medida que uno progresa en el juego, ya que en los niveles avanzados el jugador tendrá que realizar más de una acción en un mismo nivel. Cuando sea necesario realizar alguna de las acciones especiales, aparecerá un botón software indicando que se debe pulsar para realizar una acción determinada.

Por otro lado, en cada nivel se pueden encontrar, en lugares concretos y siempre fijos, hasta cien objetos coleccionables llamados “lumens”, los cuales una vez acabado el nivel determinarán la puntuación de la partida. Si se recogen todos, se obtiene una recompensa especial que, al coleccionarlas, dan acceso a nuevos niveles.

Figura II.II.IV.II Nivel de *Rayman Jungle Run*

Estos detalles son los que otorgan al título ese toque de plataformas, que caracteriza al género, y no por el contrario a un título puramente *endless run*, en el cual tanto el avance por el juego, como la recolección de ítems suele ser ilimitada.

Pese al incremento de dificultad en los niveles más avanzados, la mecánica del juego resulta ser muy intuitiva debido a la aparición de estos botones de aviso, gracias a los cuales se puede reaccionar con tiempo más que de sobra ante los eventos. Además, la velocidad no es excesivamente elevada, por lo que se le puede considerar un juego relativamente tranquilo.

Con respecto a los menús, destacar que son ligeramente complejos, ya que, al tener el juego mucha composición de niveles y extras, necesita de un amplio número de opciones.

Pese a ello la mecánica intuitiva del juego nos servirá como inspiración para la realización del nuestro propio.

Los textos del juego se encuentran escritos en español. Se necesita *Android* 2.3.3 *Gingerbread* o superior para poder ejecutar la aplicación.

II.III Conclusiones del análisis

En función de los resultados obtenidos del análisis de las aplicaciones, nos hemos percatado de varios detalles que es necesario destacar.

El primero es que el género *endless run* es bastante aceptado y disfrutado por el público, ya sea en su vertiente más purista, como en el caso de *Jetpack Joyride*, o en sus variedades, como puede ser el plataformas mezclado con *endless run*, *Rayman Jungle Run*. Este hecho pone de manifiesto que la aplicación que se pretende desarrollar puede llegar a labrarse un buen lugar en el mercado, estableciendo, en primera instancia, una base firme sobre la que comenzar a construir.

En segundo lugar, es necesario destacar la razón que ha llevado a *Ubisoft* a lanzar *Rayman Jungle Run*. Tal y como hemos mencionado previamente, ese es uno de los objetivos que se pretende conseguir con este proyecto, por lo tanto los pasos que ha dado la compañía y los éxitos cosechados por el juego se tendrán muy en cuenta durante todo el desarrollo del mismo.

Para finalizar las conclusiones extraídas sobre los videojuegos, es necesario hablar sobre la dificultad de los mismos y lo intuitivos que pueden llegar a ser. Dado que uno de los objetivos planteados es llegar al mayor número de público posible, es necesario tener en cuenta que una parte del público objetivo puede ser y será infantil, y por ende puede que no tenga experiencia con los videojuegos. Por este motivo, será necesario plantear un juego cuyo nivel de dificultad no sea muy elevando, o bien tenga varios niveles de dificultad fácilmente accesibles y distinguibles. Asimismo, un juego que cause tensión, como es el caso de *The Impossible Game*, o que requiera reacciones rápidas en el último momento, como ocurre en la saga *Temple Run*, tampoco es adecuado para cumplir con nuestro objetivo. Con respecto a los menús de juego, *Jetpack Joyride*, *Temple Run* y *Rayman Jungle Run* no cumplen con nuestros requisitos ya que son, en cierto modo, demasiado complejos para según qué público. Por el contrario, *The Impossible Game* cuenta con menús sencillos y muy intuitivos, referencia que se tendrá en cuenta a la hora de desarrollar los propios.

Capitulo III: Especificación de requisitos

El objetivo de este capítulo es realizar la especificación de requisitos software, la cual va destinada a aquellos que deseen informarse sobre la cumplimentación de requisitos de esta aplicación.

Se ha tomado como referencia el documento al respecto de *Métrica v3*, accesible desde la web del *Portal de administración electrónica del Gobierno de España* [7]. Esto ha sido así, pese a que Métrica v3 es una metodología de desarrollo pesada, puesto que consideramos que el tratamiento que da a los requisitos software es completamente correcto en su forma y adecuado para todo tipo de metodologías, ya que los requisitos son la base de cualquiera de ellas.

Este capítulo se estructura en dos secciones. En la primera se describirá la aplicación tal y como se quiere crear, para a continuación, en la segunda, realizar una especificación técnica de los requisitos que se exponen en lenguaje natural en el primer apartado.

III.I Descripción de la aplicación a desarrollar

III.I.I Contexto

El contexto en el que se engloba el desarrollo de esta aplicación es de amplia expansión tecnológica en cuanto a smartphones y tablets se refiere, que permite a todos los ciudadanos disponer de uno. Asimismo, en cuanto al campo de los videojuegos, nos encontramos sumergidos en una etapa de cambio. La llegada inminente de una nueva generación de consolas conlleva un periodo previo de calma en cuanto a novedades e innovación, ya que, la mayoría de los estudios de desarrollo de videojuegos, se centran en los futuros lanzamientos de esa nueva generación. Estos hechos, junto con la tendencia al estancamiento en el mercado clásico del desarrollo de videojuegos, nos han llevado a plantearnos la posibilidad de realizar este videojuego, trasladando los éxitos del mercado tradicional de consola a los prolíficos dispositivos móviles.

III.I.II Descripción de la aplicación

Se pretende desarrollar un videojuego de género *endless run*, portando los personajes de grandes éxitos de consola al mercado de los dispositivos móviles.

Inicialmente, y como prueba de concepto, basándonos en los siguientes videojuegos: Pokémon, Super Mario Bros y Kirby, se realizará una adaptación extrayendo los elementos necesarios para el videojuego a desarrollar.

Se pretende preservar la esencia de cada videojuego, por lo que se procurará respetar la coherencia de cada uno, alterándola lo menos posible. Esto es, cada personaje siempre se enfrentará a enemigos de su mismo videojuego y tendrá habilidades propias de sí mismo. Nunca se romperá este principio pues se rompería la coherencia del juego.

El objetivo del juego será aguantar lo máximo posible corriendo sin morir. Cuanto más tiempo se aguante mayor puntuación se conseguirá, pudiéndose recoger monedas durante la partida para incrementar la puntuación.

Se procurará que los menús del juego sean lo más intuitivos posibles, con el fin de completar el objetivo planteado de alcanzar el mayor público posible.

III.I.III Funciones de la aplicación

Las funciones que tendrá la aplicación son las siguientes:

- Iniciar partida.

- Seleccionar personaje.
- Salir del juego.
- Finalizar partida.
- Almacenar records de cada personaje.
- Cargar records de cada personaje.
- Reiniciar partida.
- Reproducir música.
- Reproducir sonidos FX.
- Responder al control del protagonista.
- Dibujar los elementos en pantalla.
- Actualizar los estados de los elementos.
- Reciclado de memoria.
- Control de los frames por segundo (FPS).
- Posibilidad de silenciar la música.
- Posibilidad de silenciar los sonidos FX.
- Posibilidad de cambiar a modo clásico de música.
- Posibilidad de cambiar el modo de control.

III.I.IV Características del usuario objetivo

Los usuarios de esta aplicación deberán cumplir el requisito de poseer un smartphone o tablet *Android*.

El margen de edad del usuario no está definido con claridad, ya que se pretende llegar al máximo número de público posible, pero se espera que la aplicación tenga mayor afinidad con usuarios nacidos a partir de los años 80, ya que la ambientación del juego incluirá personajes reconocidos de esa época.

III.I.V Restricciones de diseño

La aplicación será desarrollada con el nivel de API objetivo 17, correspondiente a la versión *4.2 Jelly Bean* de *Android*. No obstante, deberá asegurarse que la aplicación pueda ser ejecutada en el mayor número de versiones posibles, reduciendo por tanto el nivel mínimo de API necesaria y controlando, si fuese necesario, la correcta ejecución de la aplicación en todas las versiones.

El lenguaje de programación utilizado durante todo el desarrollo será *Java*, siendo este el lenguaje indicado para la programación de aplicaciones *Android*.

Para la correcta instalación de la aplicación se deberá disponer de suficiente espacio de almacenamiento libre.

La velocidad de ejecución de la aplicación depende, en última instancia, del dispositivo en el que se ejecute. No obstante, la aplicación se optimizará para que esta limitación sea lo menos restrictiva posible.

En cuanto a lo que respecta a los permisos de *Android*, se considera la política de seguridad "todo lo que no está permitido, está prohibido".

III.II Tablas de requisitos

En esta sección se incluye la descripción de los requisitos de la aplicación a desarrollar en formato de tabla. Mediante esta especificación de requisitos se pretende dar a conocer a los diseñadores las restricciones que el desarrollo del proyecto ha de respetar, así como proporcionar al equipo de pruebas la suficiente información como para realizar un catálogo de pruebas completo para asegurar el cumplimiento de todos los requisitos.

Es necesario mencionar, como ya se hizo previamente, que el proceso asociado a la extracción de requisitos se ha basado en el *Proceso de Análisis del Sistema de la Información* de *Métrica v3*[6].

En la sección III.II.I se incluirán los requisitos funcionales de la aplicación, es decir, aquellos que determinen la funcionalidad que la aplicación es capaz de desarrollar.

En la sección III.II.II se incluirán los requisitos no funcionales de la aplicación, es decir, aquellos que no se refieren directamente a la funcionalidad proporcionada sino, más bien, a las propiedades emergentes de la misma. En este apartado diferenciaremos entre requisitos de diseño, requisitos de rendimiento y requisitos de interfaz.

Para definir correctamente los requisitos se utilizará una tabla que contendrá la información de cada requisito acorde al siguiente modelo:

ID	Identificador	**Nombre**	Nombre del requisito
Prioridad	Alta/Media/Baja	**Repercusión**	Alta/Media/Baja
Dependencias	Otro identificador	**Fecha**	dd/mm/aaaa
Descripción	Descripción clara y concisa del requisito así como la información necesaria para su posterior codificación.		

Tabla III.II.I Modelo de tabla de requisitos

[6] Análisis del Sistema de la Información, Métrica v 3: http://administracionelectronica.gob.es/recursos/pae_000001030.pdf

A continuación se describe cada uno de los apartados de la tabla y el formato que siguen en el caso de ser necesario:

- **Identificador**: cada requisito software incluirá una identificación para facilitar su traza. Estará compuesto por RT[S]-XXX donde T será F para los requisitos funcionales y NF para los no funcionales, los cuales a su vez incluirán el campo opcional S que podrá ser D, para los requisitos de diseño, R para los de rendimiento e I para los de interfaz. Para terminar se incluirá un valor numérico XXX, comenzando en 001, incrementado en uno por cada nuevo requisito diferenciando entre los dos tipos principales.
- **Nombre**: identifica el requisito de software por medio de una breve descripción.
- **Prioridad**: cualidad que indica la urgencia e importancia del requisito de cara al desarrollador. Puede ser alta, media o baja.
- **Repercusión**: cualidad que indica la relevancia, importancia y repercusión de una modificación en ese requisito en el proyecto.
- **Dependencias**: relaciones con otros requisitos. Constará de uno o varios identificadores en el caso de existir dependencias o de ninguno en el caso contrario.
- **Fecha**: fecha en la cual se redactó el requisito por primera vez.
- **Descripción:** descripción ampliada del requisito e información que se quiere transmitir al desarrollador con respecto al mismo.

III.II.I Requisitos funcionales

ID	RF-001	**Nombre**	Selección de personaje
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación permitirá seleccionar un personaje. Se mostrará una pantalla con todos los personajes disponibles permitiendo la selección entre ellos.		

Tabla III.II.I.I Requisito funcional RF-001: Selección de personaje

ID	RF-002	**Nombre**	Ciclo de vida
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación tendrá en cuenta el ciclo de vida de las actividades en *Android* y responderá coherentemente.		

Tabla III.II.I.II Requisito funcional RF-002: Ciclo de vida

ID	RF-003	**Nombre**	Salida de la aplicación
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-002	**Fecha**	10/05/2013
Descripción	La aplicación permitirá salir en cualquier momento. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde al mismo.		

Tabla III.II.I.III Requisito funcional RF-003: Salida de la aplicación

ID	RF-004	**Nombre**	Reproducción de audio
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-002, RF-003	**Fecha**	10/05/2013
Descripción	La aplicación permitirá reproducir audio como parte de la banda sonora del juego. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde a ello.		

Tabla III.II.I.IV Requisito funcional RF-004: Reproducción de audio

ID	RF-005	**Nombre**	Reproducción de sonidos FX
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-002, RF-003	**Fecha**	10/05/2013
Descripción	La aplicación permitirá reproducir sonidos FX como parte de la banda sonora del juego. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde a ello.		

Tabla III.II.I.V Requisito funcional RF-005: Reproducción de sonidos FX

ID	RF-006	**Nombre**	Reproducción simultánea
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-004, RF-005	**Fecha**	10/05/2013
Descripción	La aplicación permitirá reproducir ambos tipos de sonidos simultáneamente.		

Tabla III.II.I.VI Requisito funcional RF-006: Reproducción simultánea

ID	RF-007	**Nombre**	Inicio de partida
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-001	**Fecha**	10/05/2013
Descripción	La aplicación permitirá iniciar una nueva partida siempre que el usuario quiera.		

Tabla III.II.I.VII Requisito funcional RF-007: Inicio de partida

ID	RF-008	**Nombre**	Reinicio de partida
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-001	**Fecha**	10/05/2013
Descripción	La aplicación permitirá reiniciar la partida una vez el jugador haya muerto. Para ello se deberá tener en cuenta el personaje escogido previamente.		

Tabla III.II.I.VIII Requisito funcional RF-008: Reinicio de partida

ID	RF-009	**Nombre**	Desactivar audio
Prioridad	Media	**Repercusión**	Baja
Dependencias	RF-004	**Fecha**	10/05/2013
Descripción	La aplicación permitirá desactivar la reproducción de audio. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde a ello.		

Tabla III.II.I.IX Requisito funcional RF-009: Desactivar audio

ID	RF-010	**Nombre**	Desactivar sonidos FX
Prioridad	Media	**Repercusión**	Baja
Dependencias	RF-005	**Fecha**	10/05/2013
Descripción	La aplicación permitirá desactivar la reproducción de sonidos FX. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde a ello.		

Tabla III.II.I.X Requisito funcional RF-010: Desactivar sonidos FX

ID	RF-011	**Nombre**	Modo de audio
Prioridad	Baja	**Repercusión**	Baja
Dependencias	RF-004, RF-009	**Fecha**	21/05/2013
Descripción	La aplicación permitirá alterar entre modo clásico y modo normal a la hora de reproducir audio como parte de la banda sonora del juego.		

Tabla III.II.I.XI Requisito funcional RF-011: Modo de audio

ID	RF-012	**Nombre**	Cambio a control por sensor
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	30/05/2013
Descripción	La aplicación permitirá cambiar el modo de control clásico al del sensor de movimiento. Para ello se deberá tener en cuenta el ciclo de vida de las actividades en *Android* y responder acorde a ello.		

Tabla III.II.I.XII Requisito funcional RF-012: Cambio a control por sensor

ID	RF-013	Nombre	Almacenamiento de records
Prioridad	Media	Repercusión	Baja
Dependencias	-	Fecha	30/05/2013
Descripción	La aplicación almacenará las puntuaciones más altas de cada personaje, de tal forma que cuando se vuelva a iniciar el juego se tenga disponible este valor.		

Tabla III.II.I.XIII Requisito funcional RF-013: Almacenamiento de records

ID	RF-014	Nombre	Dibujado de entorno
Prioridad	Alta	Repercusión	Media
Dependencias	-	Fecha	10/05/2013
Descripción	La aplicación dibujará, a cada iteración del ciclo básico del juego, todos los componentes del juego.		

Tabla III.II.I.XIII Requisito funcional RF-014: Dibujado de entorno

ID	RF-015	Nombre	Actualización del entorno
Prioridad	Alta	Repercusión	Media
Dependencias	RF-014	Fecha	10/05/2013
Descripción	La aplicación actualizará los estados, a cada iteración del ciclo básico del juego, de todos los componentes del juego.		

Tabla III.II.I.XIII Requisito funcional RF-015: Actualización del entorno

ID	RF-016	**Nombre**	Comprobación de colisión con bonificadores
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015	**Fecha**	10/05/2013
Descripción	La aplicación controlará las posibles colisiones del personaje con los bonificadores del entorno, eliminará el bonificador y asignará al personaje la mejora correspondiente si esto sucede.		

Tabla III.II.I.XIII Requisito funcional RF-016: Comprobación de colisión con bonificadores

ID	RF-017	**Nombre**	Comprobación de colisiones con monedas
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015	**Fecha**	10/05/2013
Descripción	La aplicación controlará las posibles colisiones del personaje con las monedas del entorno, las eliminará e incrementará el contador de monedas si esto sucede.		

Tabla III.II.I.XIII Requisito funcional RF-017: Comprobación de colisiones con monedas

ID	RF-018	**Nombre**	Comprobación de colisiones con enemigos
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014	**Fecha**	10/05/2013
Descripción	La aplicación controlará las posibles colisiones del personaje con los enemigos del entorno y terminará el juego si esto sucede.		

Tabla III.II.I.XIII Requisito funcional RF-018: Comprobación de colisiones con enemigos

ID	RF-019	**Nombre**	Generación pseudoaleatoria de bonificadores
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	10/05/2013
Descripción	La aplicación generará nuevos bonificadores pseudoaleatoriamente.		

Tabla III.II.I.XIII Requisito funcional RF-019: Generación pseudoaleatoria de bonificadores

ID	RF-020	**Nombre**	Generación pseudoaleatoria de monedas
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	10/05/2013
Descripción	La aplicación generará nuevas monedas pseudoaleatoriamente.		

Tabla III.II.I.XIII Requisito funcional RF-020: Generación pseudoaleatoria de monedas

ID	RF-021	**Nombre**	Generación pseudoaleatoria de enemigos
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	10/05/2013
Descripción	La aplicación generará nuevos enemigos pseudoaleatoriamente.		

Tabla III.II.I.XIII Requisito funcional RF-021: Generación pseudoaleatoria de enemigos

ID	RF-022	**Nombre**	Generación de suelo
Prioridad	Baja	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	30/05/2013
Descripción	La aplicación generará nuevas unidades de suelo a cada ciclo de ejecución.		

Tabla III.II.I.XIII Requisito funcional RF-022: Generación de suelo

ID	RF-023	**Nombre**	Generación pseudoaleatoria de nubes
Prioridad	Baja	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	30/05/2013
Descripción	La aplicación generará nuevas nubes pseudoaleatoriamente.		

Tabla III.II.I.XIII Requisito funcional RF-023: Generación pseudoaleatoria de nubes

ID	RF-024	**Nombre**	Reciclado de memoria
Prioridad	Alta	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	10/05/2013
Descripción	La aplicación reciclará memoria eliminando los objetos cuyas coordenadas se sitúen fuera de la pantalla por la izquierda del personaje.		

Tabla III.II.I.XIII Requisito funcional RF-024: Reciclado de memoria

ID	RF-025	**Nombre**	Visualización de animaciones
Prioridad	Media	**Repercusión**	Media
Dependencias	RF-014, RF-015, RNFR-002	**Fecha**	10/05/2013
Descripción	La aplicación mostrará las animaciones de los elementos que las contengan. Estos elementos serán los personajes, los enemigos, los bonificadores y las monedas.		

Tabla III.II.I.XIII Requisito funcional RF-025: Visualización de animaciones

ID	RF-024	**Nombre**	Control del personaje
Prioridad	Alta	**Repercusión**	Media
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación permitirá el control del personaje respondiendo con un salto a la interacción del usuario.		

Tabla III.II.I.XIII Requisito funcional RF-024: Control del personaje

III.II.II Requisitos no funcionales

ID	RNFR-001	**Nombre**	Tiempo de respuesta
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación controlará los tiempos de respuesta asociados a las acciones del jugador, de tal forma que la respuesta sea prácticamente instantánea.		

Tabla III.II.II.I Requisito no funcional de rendimiento RNFR-001: Tiempo de respuesta

ID	RNFR-002	**Nombre**	Frecuencia de refresco
Prioridad	Alta	**Repercusión**	Media
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación respetará la frecuencia de refresco asignada, que por defecto no será menor a 15 FPS, de tal forma que se mantenga el efecto de persistencia de la visión.		

Tabla III.II.II.I Requisito no funcional de rendimiento RNFR-002: Frecuencia de refresco

ID	RNFD-003	**Nombre**	Coherencia
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación respetará la coherencia existente con los personajes a utilizar.		

Tabla III.II.II.I Requisito no funcional de diseño RNFD-003: Coherencia

ID	RNFD-004	Nombre	Menús intuitivos
Prioridad	Alta	Repercusión	Baja
Dependencias	-	Fecha	10/05/2013
Descripción	Los menús de la aplicación serán lo más intuitivos posibles de tal forma que su funcionalidad sea la que se infiere en primera instancia y no quepa lugar a dudas.		

Tabla III.II.II.I Requisito no funcional de diseño RNFD-004: Menús intuitivos

ID	RNFI-005	Nombre	Control táctil
Prioridad	Alta	Repercusión	Baja
Dependencias	RF-024	Fecha	10/05/2013
Descripción	El control del juego responderá a las acciones realizadas sobre la pantalla táctil.		

Tabla III.II.II.I Requisito no funcional de interfaz RNFI-005: Control táctil

ID	RNFI-006	Nombre	Control por sensor
Prioridad	Alta	Repercusión	Baja
Dependencias	RF-024	Fecha	30/05/2013
Descripción	El control del juego responderá a las acciones realizadas a través del sensor de movimiento del dispositivo.		

Tabla III.II.II.I Requisito no funcional de interfaz RNFI-006: Control por sensor

ID	RNFD-007	**Nombre**	Compatibilidad con dispositivos
Prioridad	Alta	**Repercusión**	Baja
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación será compatible con el mayor número de versiones de *Android* posible.		

Tabla III.II.II.I Requisito no funcional de diseño RNFD-007: Compatibilidad con dispositivos

ID	RNFD-008	**Nombre**	Compatibilidad con resoluciones
Prioridad	Alta	**Repercusión**	Baja
Dependencias	RF-024	**Fecha**	10/05/2013
Descripción	La aplicación será compatible con diferentes resoluciones de pantalla.		

Tabla III.II.II.I Requisito no funcional de diseño RNFD-008: Compatibilidad con resoluciones

ID	RNFR-009	**Nombre**	Compatibilidad con formatos de audio
Prioridad	Media	**Repercusión**	Media
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación será compatible con diferentes formatos de audio como mp3, ACC, MIDI y WAV.		

Tabla III.II.II.I Requisito no funcional de rendimiento RNFR-009: Compatibilidad con formatos de audio

ID	RNFI-010	**Nombre**	Textos legibles
Prioridad	Alta	**Repercusión**	Media
Dependencias	-	**Fecha**	10/05/2013
Descripción	La aplicación se adaptará a las diferentes resoluciones de pantalla generando siempre textos legibles.		

Tabla III.II.II.I Requisito no funcional de interfaz RNFI-010: Textos legibles

Capítulo IV: Diseño de la aplicación

El objetivo de este capítulo es presentar el *prototipo* de la aplicación que se generó durante las primeras fases de conceptualización, el cual se ha visto modificado posteriormente en las fases de diseño e implementación. El objetivo de los prototipos es el de aproximar gráficamente la solución de tal forma que pueda ser compartida con el cliente para trabajar sobre ella antes de comenzar a realizar la aplicación en sí, ya que de este modo resulta más fácil corregir fallos que en fases más avanzadas del proyecto. En nuestro caso, y dado que no era necesario mostrarlo ante el cliente, se realizó el prototipo para tener una idea previa básica sobre el aspecto final que presentaría la aplicación. Posteriormente se trabajó sobre este prototipo con el fin de refinar los detalles de la misma.

Una vez presentado el prototipo se analizarán las alternativas de diseño que se encontraban disponibles y se expondrán los motivos que llevaron a decidirse por la escogida.

Para terminar con el capítulo, se presentarán los diagramas de clases y de secuencia, de modo que el lector pueda tener una perspectiva más visual del diseño llevado a cabo en este proyecto.

IV.I Prototipo

En esta sección se muestra el prototipo de la aplicación realizado durante la primera semana de Febrero de 2013.

Cabe destacar que, pese a que se supone que el resultado final será distinto del estado inicial mostrado por el prototipo, el poder ver en una imagen el aspecto que se esperaba tuviese la aplicación sirvió de gran ayuda para hacernos una idea global del funcionamiento de la misma.

IV.I.I Menú principal

Esta pantalla de muestra está compuesta por una imagen de fondo, la cual se representa por el fondo blanco, el título de la aplicación y dos posibles opciones, el botón de Inicio y el de Opciones.

La aplicación se iniciará en este menú, desde donde el usuario podrá iniciar fácilmente una partida o bien modificar las opciones de la aplicación para adaptarla a su gusto.

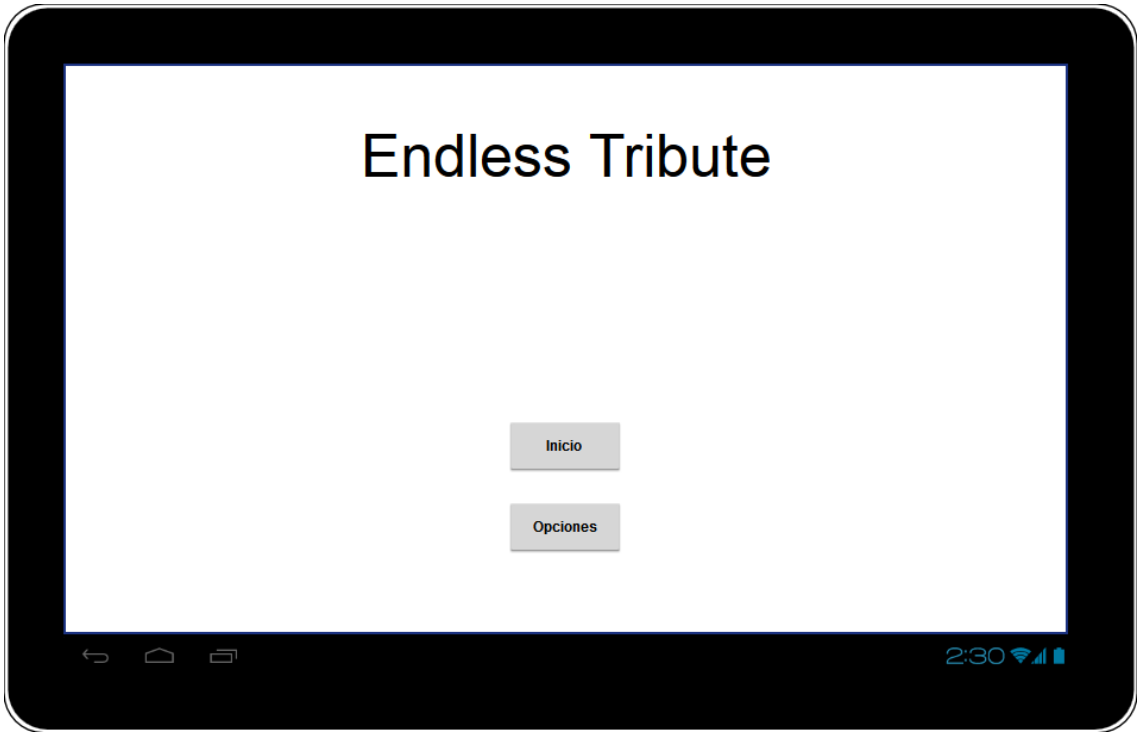


Figura IV.I.I.I Prototipo de menú principal

IV.I.II Menú de opciones

Desde esta ventana el usuario podrá cambiar las diferentes opciones de la aplicación. Estas opciones se presentarán como checkboxes con valor on/off en el formato más sencillo posible, de tal forma que pueda ser entendido fácilmente y de forma intuitiva.

El usuario podrá retroceder al menú principal desde esta ventana en cualquier momento pulsando la tecla atrás en el dispositivo.

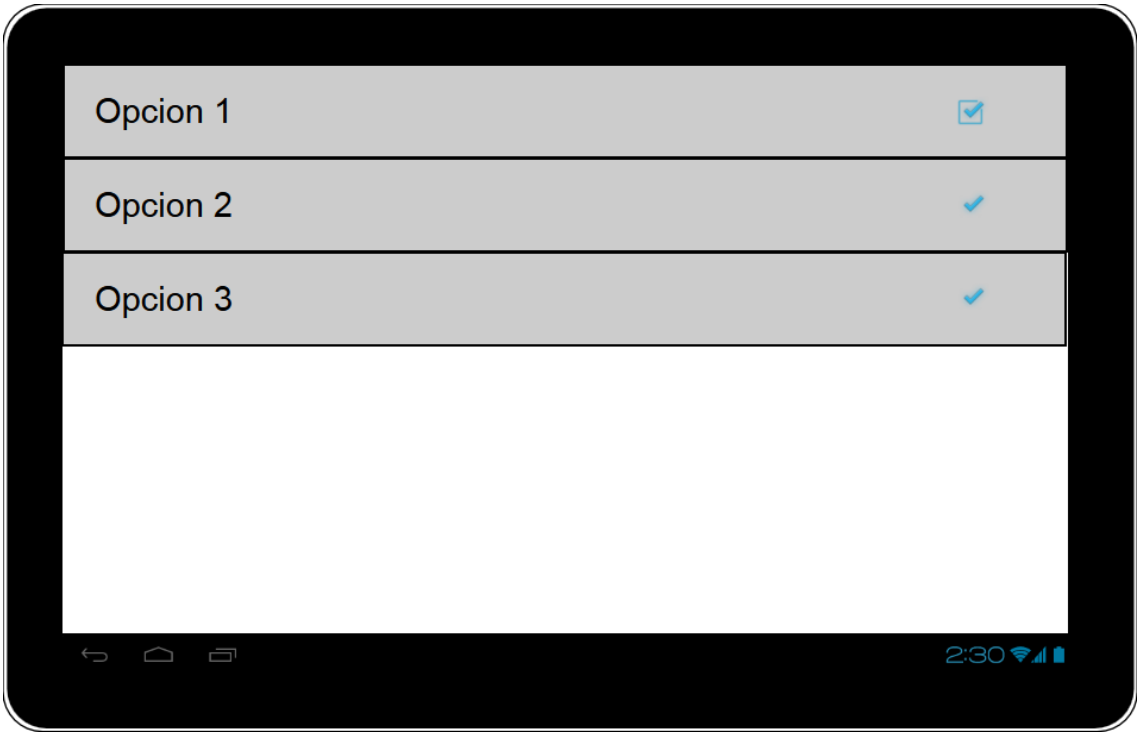


Figura IV.I.II.I Prototipo de menú de opciones

IV.I.III Menú de selección de personaje

En esta ventana se mostrarán los personajes disponibles, entre los cuales el usuario pueda escoger previamente a comenzar el juego.

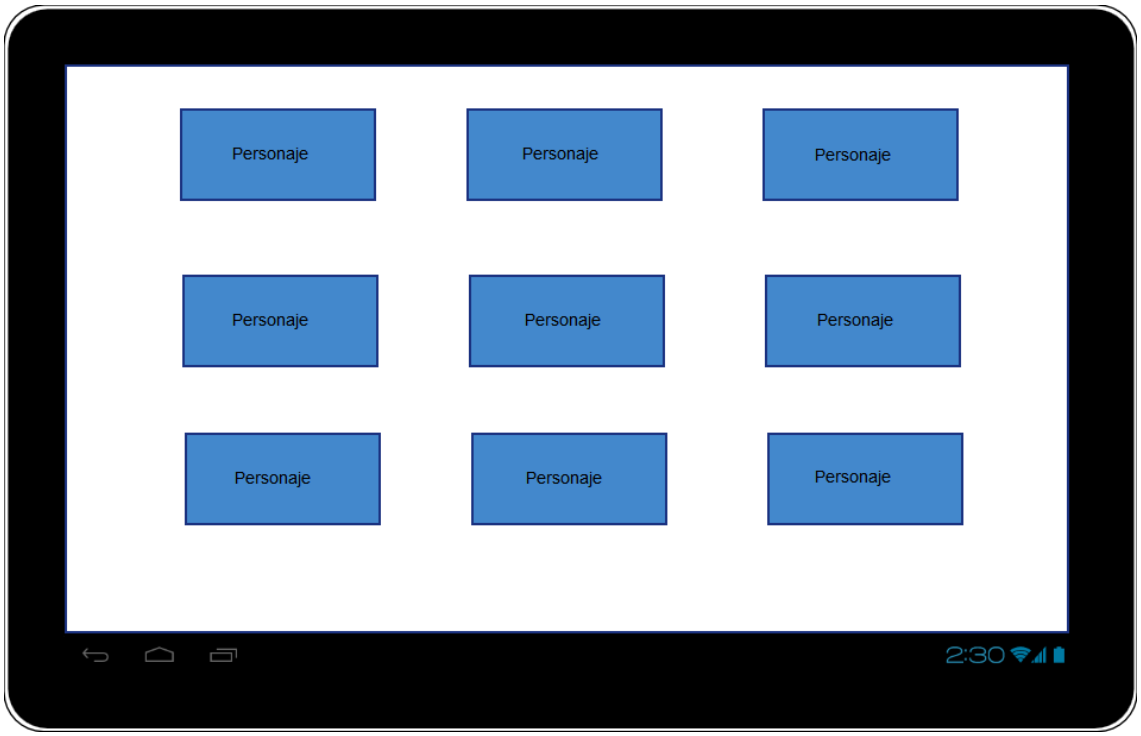


Figura IV.I.III.I Prototipo de menú de selección de personaje

IV.I.IV Pantalla de juego

En esta ventana se muestra el juego en ejecución. Se pueden observar a modo de prototipo las monedas, el jugador, un enemigo, un power up y el suelo. Asimismo, se muestra la puntuación, las monedas recogidas y el récord actual.

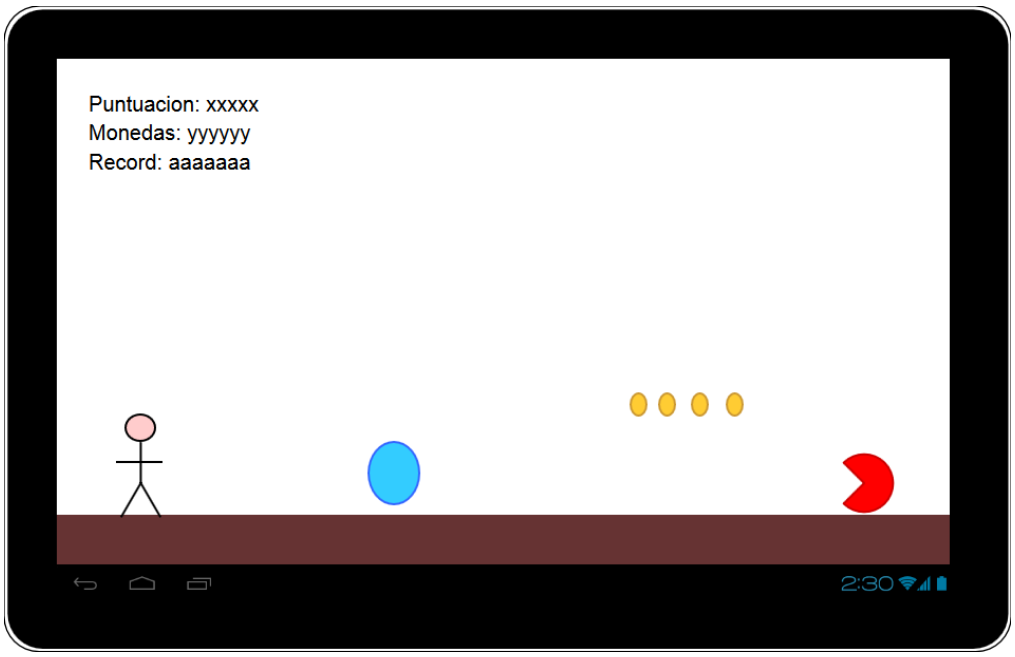


Figura IV.I.IV.I Prototipo de pantalla de juego 1

Una vez la partida ha terminado por haber muerto el jugador se mostrará una opción permitiendo reiniciar el juego y los resultados de la partida.

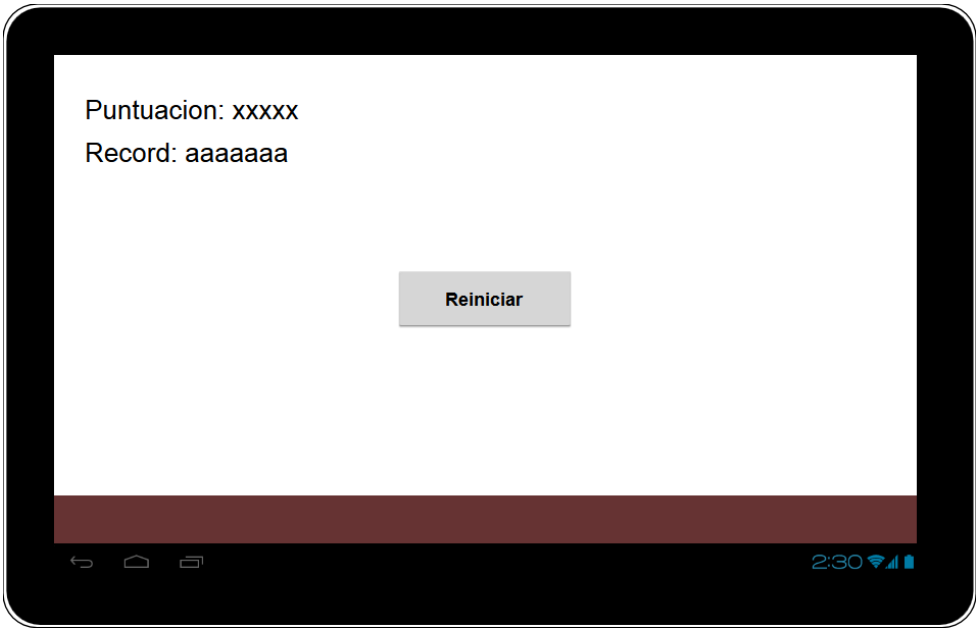


Figura IV.I.IV.II Prototipo de pantalla de juego 2

IV.II Alternativas de diseño

En esta sección se exponen las alternativas encontradas durante la fase de diseño y cuál ha sido escogida, así como los motivos que llevaron a ello.

A la hora de plantearnos el cómo llevar a cabo la aplicación, lo primero que se nos vino a la cabeza fue que, al no tener experiencia previa con el desarrollo de videojuegos, tampoco se tenía ningún tipo de conocimiento con respecto al cómo se hacían. Tampoco se había experimentado nunca antes con aplicaciones de videojuegos en dispositivo móvil, por lo que las opciones para desarrolladores que éstos proporcionaban nos eran completamente desconocidas.

Para comenzar con la investigación sobre cómo desarrollar un videojuego se decidió informarse acerca de qué era necesario y tras visitar varias páginas web el concepto de *motor de desarrollo de videojuegos* se hizo cada vez más habitual.

Un *motor de desarrollo de videojuegos* consiste en una serie de rutinas (o métodos) que facilitan el diseño, la creación y la representación de un juego [8]. Habitualmente, un motor de desarrollo implementa un framework, que se encarga de facilitar el acceso a determinadas funciones necesarias mediante un nivel de software intermedio. Las diferentes interfaces que permiten el acceso a estas funciones se denominan *APIs*.

Por lo tanto, una vez llegados a este punto se tuvo que investigar acerca de qué motores de desarrollo existían para desarrollar en *Android*. De nuevo, una vez más, fácilmente se encontró que, actualmente existen gran variedad de motores de desarrollo para Android, entre los cuales podemos citar *Rokon*, *UNITY* y *Android-2D-Engine* como motores y *Libgdx* como framework.

Tras evaluar las características que proporcionaba cada uno y valorar el balance entre funcionalidad y curva de aprendizaje se optó por no utilizar ni motor de desarrollo ni framework. Los motivos que llevaron a tomar esta decisión son los siguientes:

- Al no tener amplios conocimientos de *Android*, utilizar un motor que enmascarase la parte de *Android*, aunque fuese en pos de facilitar las tareas, no se consideró apropiado, puesto que una de las metas personales marcadas es comprender más este sistema operativo.

- Utilizar un motor de videojuegos concreto te limita en cierta medida a aprender sobre un tema concreto. El objetivo de este trabajo no es enfocarse en un aspecto concreto del desarrollo de videojuegos, como es la programación con un motor, sino comprender a gran escala el proceso de desarrollo de los videojuegos.
- Dado que nuestro objetivo principal es realizar una aplicación para el mayor público posible, es decir, una aplicación sencilla e intuitiva, no se necesitará la potencia que un motor proporciona.

Por estos y otros motivos personales que no sería adecuado explicar en un documento de esta índole, se decidió realizar la aplicación sin la ayuda de un motor de desarrollo.

IV.III Diagramas de clases

Tras la realización del prototipo y de la elección entre todas las alternativas de diseño se optó por realizar un breve diagrama de clases con el fin de tener un punto de vista más técnico previamente a la implementación en sí de la aplicación.

Tras las continuas iteraciones siguiendo las bases de nuestra metodología, el diseño de la aplicación ha cambiado y por lo tanto el estado inicial del diagrama de clases también. Con el fin de ser lo más realistas y objetivos posibles se ha optado por mostrar el estado final de los diagramas de clases. De este modo, podremos mostrar el estado más actual de la aplicación, ya que inicialmente los diagramas que se propusieron eran poco eficientes.

Los diagramas muestran los atributos y métodos que posee cada clase, así como las relaciones entre ellas. Estas últimas sólo serán representadas si ambas clases de la relación forman parte de la aplicación ya que, por la naturaleza de *Java*, casi todas las clases implementan o heredan de otras, por lo que los diagramas se convertirían en algo intratable y con poco sentido.

La aplicación se encuentra separada por paquetes, relación que mantendremos a la hora de describir los diagramas en pos de una mejor organización.

A continuación se muestra un diagrama de clases que relaciona los diferentes paquetes que contiene la aplicación previamente a describir cada una de las clases.

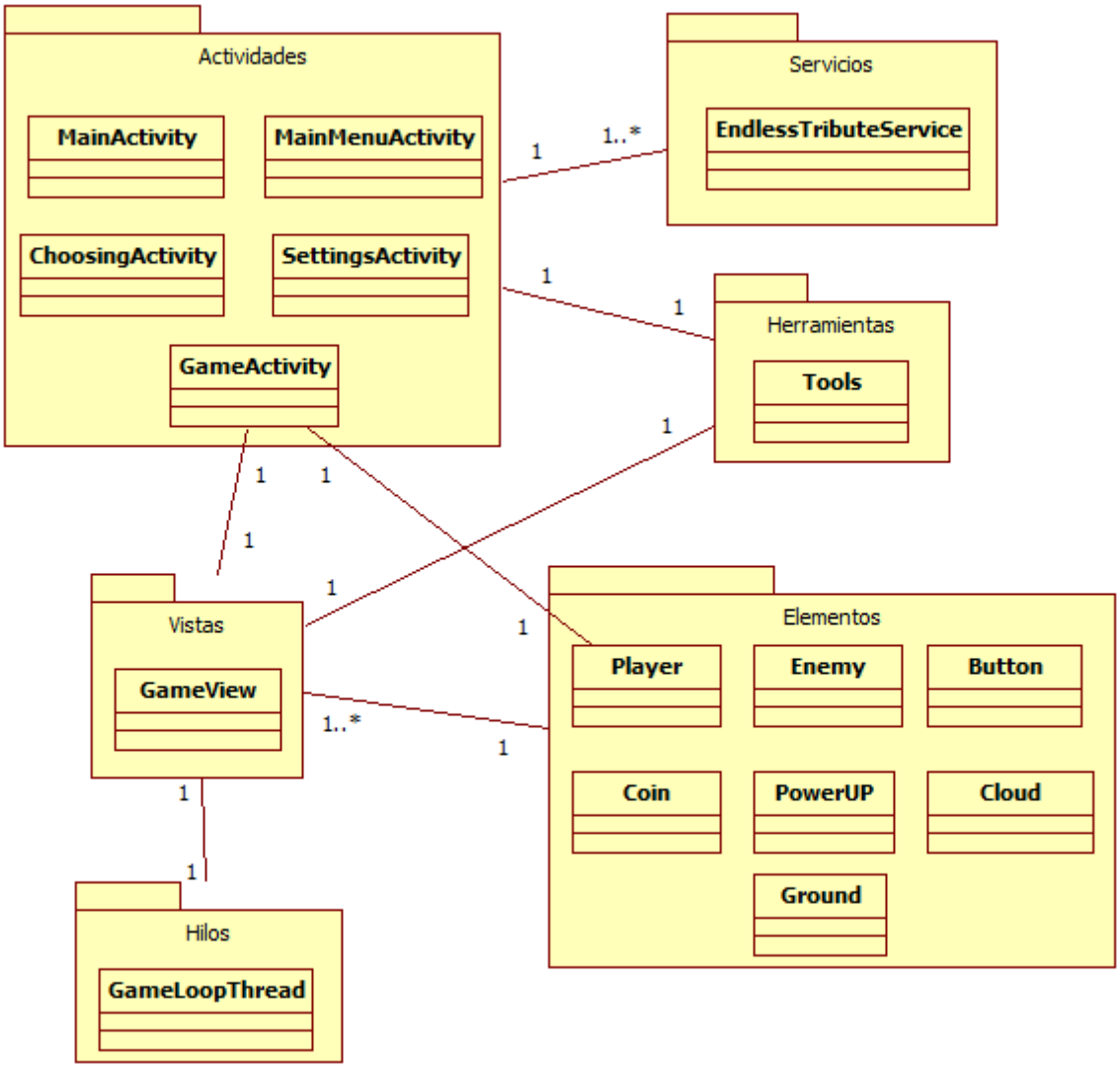


Figura IV.III.I Diagrama de clases de la aplicación

IV.III.I Actividades

En esta sección se explicarán todas las clases que heredan de *Activity* y que por tanto son las clases que forman la estructura de la aplicación.

La clase *MainActivity* (figura IV.III.I.I) se encarga de iniciar la aplicación. Para ello inicia el servicio *EndlessTributeService* el cual se utilizará constantemente durante toda la aplicación para reproducir música y sonidos FX. Además, mientras el servicio se está iniciando la actividad mostrará una animación previamente a lanzar la llamada a la siguiente actividad. Una vez hecho esto, la actividad se cerrará imposibilitando que el usuario pueda volver a acceder a ella.

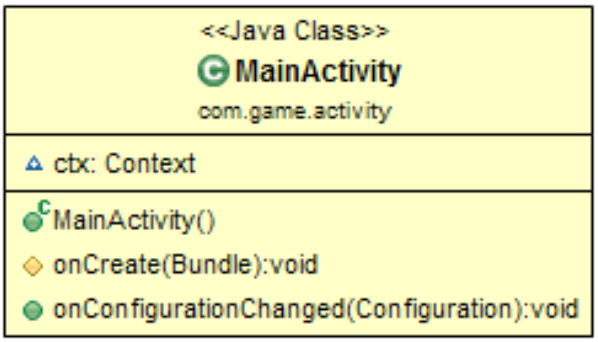


Figura IV.III.I.I Diagrama de clase *MainActivity*

La actividad a la que *MainActivity* llama antes de cerrarse es *MainMenuActivity* (figura IV.III.I.II). Esta es la actividad principal de la aplicación desde la que se accede al resto de actividades. Inicialmente la actividad creará un fondo con los botones de *Inicio* y *Opciones*, a elegir aleatoriamente entre varios disponibles, creados al estilo de algunos personajes del juego. Asimismo, la clase implementa la interfaz *OnTouchListener*, de tal forma que pueda detectar cuándo se realizan pulsaciones en la pantalla y responder acorde al lugar en el que se haya realizado la pulsación. Esta clase se conecta al servicio *EndlessTributeService* durante su creación y posteriormente compartirá esta conexión con el resto de las actividades. Además, esta clase calculará los valores de altura y anchura de la pantalla del dispositivo en el que se esté ejecutando la aplicación, los cuales posteriormente serán utilizados tanto por ella como por el resto de las actividades.

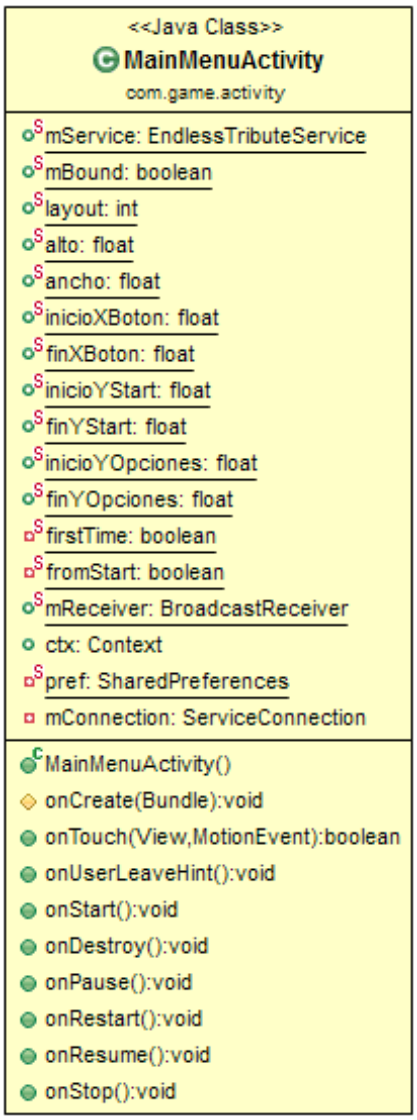


Figura IV.III.I.II Diagrama de clase *MainMenuActivity*

Al pulsar el botón *Opciones*, la actividad *MainMenuActivity* llamará a la clase *SettingsActivity* (figura IV.III.I.III), creándose una instancia de la misma.

La clase *SettingsActivity* hereda de *PreferenceActivity*, clase proporcionada por *Android* que facilita la gestión de preferencias de aplicaciones. En esta clase se podrán activar o desactivar diversas opciones, actualizando el estado de la aplicación en tiempo real.

Una vez el usuario ha terminado de asignar sus preferencias puede pulsar la tecla atrás del dispositivo para volver a *MainMenuActivity*.

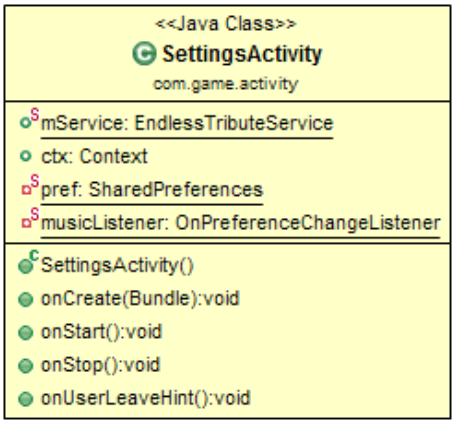


Figura IV.III.I.III Diagrama de clase *SettingsActivity*

Al pulsar en el botón Inicio la actividad *MainMenuActivity* llamará a la clase *ChoosingActivity* (figura IV.III.I.IV), creándose una instancia de la misma.

La clase *ChoosingActivity* es la encargada de mostrar los personajes disponibles a elegir. Para ello, carga el fondo de pantalla que había sido seleccionado previamente en *MainMenuActivity* y sobre él muestra una imagen por cada uno de los personajes disponibles. Cada una de estas imágenes tendrá la función de iniciar una nueva partida con el personaje escogido. La clase implementa *OnTouchListener*, al igual que *MainMenuActivity*, para poder recibir y gestionar los toques en la pantalla.

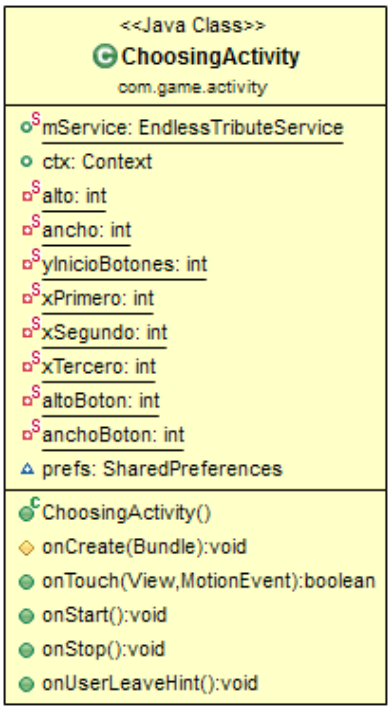


Figura IV.III.I.IV Diagrama de clase *ChoosingActivity*

Una vez seleccionado un personaje, *ChoosingActivity* llamará a *GameActivity* (figura IV.III.I.V), la actividad en la cual se aloja la vista que controla toda la lógica del juego.

Esta actividad implementa *SensorEventListener*, interfaz que define los métodos necesarios para capturar y poder tratar eventos de los sensores del dispositivo. Concretamente, se van a tratar los eventos identificados como provenientes del sensor de movimiento o acelerómetro.

En el momento de creación de esta actividad se inicia la vista pasando los parámetros del contexto, tamaño de la pantalla, jugador escogido y opciones. Por último, asigna la vista a la propia actividad. La principal función de esta actividad es la de recibir los eventos del sensor, cuando este modo de juego se encuentra activado, y transferir la orden de actuar a la vista si el movimiento se encuentra dentro de las acciones seleccionadas para ello.

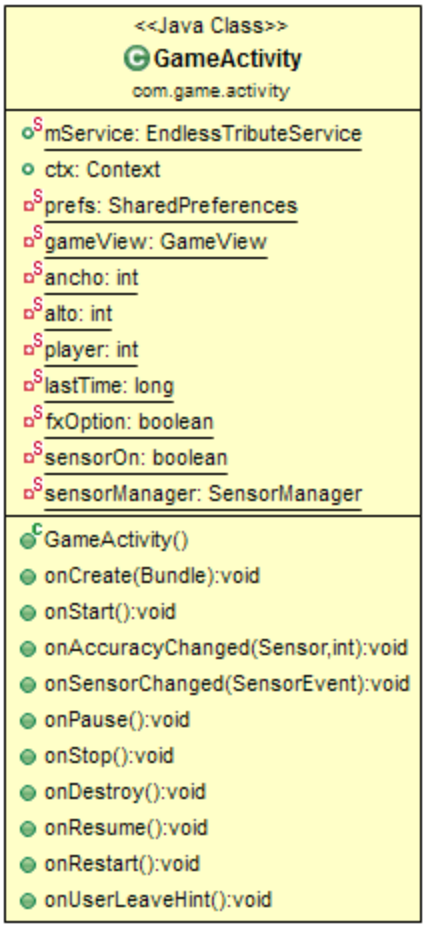


Figura IV.III.I.V Diagrama de clase *GameActivity*

IV.III.II Vistas

La aplicación únicamente posee una clase definida como vista y esta es la clase en la que se alojan todas las funciones que dan lugar al juego en sí. Con ayuda de *GameLoopThread* y *GameActivity*, *GameView* (figura IV.III.II.I) es capaz de hacer funcionar toda la lógica del juego, lo que incluye, actualización de estados, dibujo en pantalla, comprobación de colisiones y reciclado de memoria.

Figura IV.II.II.I Diagrama de clase *GameView*

IV.III.III Hilos

Solo es necesaria la existencia de un hilo de ejecución *GameLoopThread* (figura IV.III.III.I), que se crea y se borra cada vez que se entra o sale de *GameActivity*.

Este hilo es el encargado de comprobar que se cumplen los criterios de FPS y actuar en consecuencia, así como de realizar las consecutivas llamadas al método *onDraw* de la clase *GameView* con el fin de reproducir el juego.

En el caso de que exista algún retraso extremadamente largo en la ejecución normal del bucle, el hilo realiza una serie de actualizaciones al estado de los objetos sin renderizar.

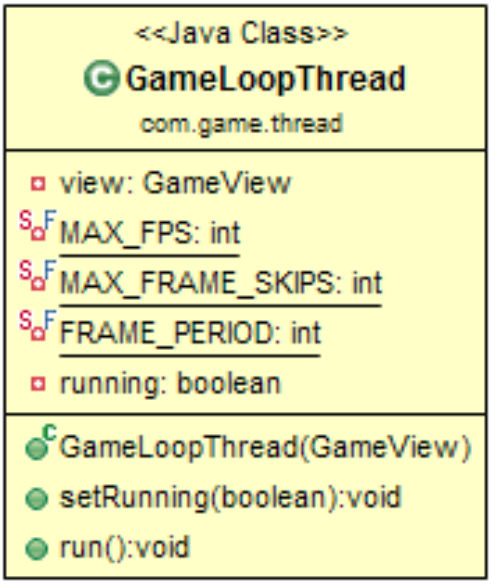


Figura IV.III.III.I Diagrama de clase *GameLoopThread*

IV.III.IV Servicios

La aplicación cuenta con un único servicio, *EndlessTributeService* (figura IV.III.IV.I), el cual se encuentra en ejecución siempre que la aplicación esté iniciada.

Este servicio se inicia en *MainActivity* y se vincula posteriormente a *MainMenuActivity*, desde donde el resto de actividades acceden a él.

Las funciones que proporciona este servicio a la aplicación son las de reproducción de música y sonidos FX.

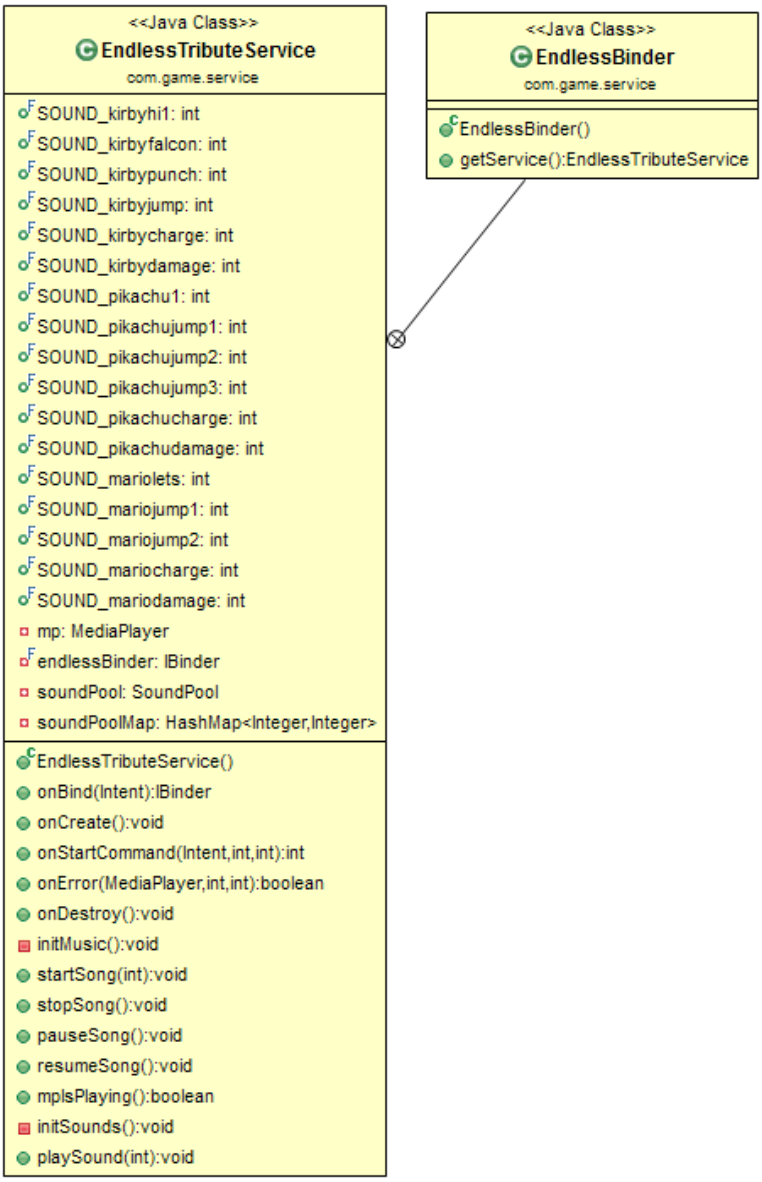


Figura IV.III.IV.I Diagrama de clase *EndlessTributeService* y *EndlessBinder*

Además, el servicio incluye la implementación de *EndlessBinder*, el cual realiza las funciones necesarias para enlazar el servicio a una actividad.

IV.III.V Elementos

Consideramos elementos a todas las clases que generan objetos clásicos de *Java*, los cuales representan entidades dentro del juego.

Estos elementos pueden ser:

- *Player* (figura IV.III.V.I): personaje protagonista, actualmente a elegir entre *Kirby*, *Mario* y *Pikachu*.

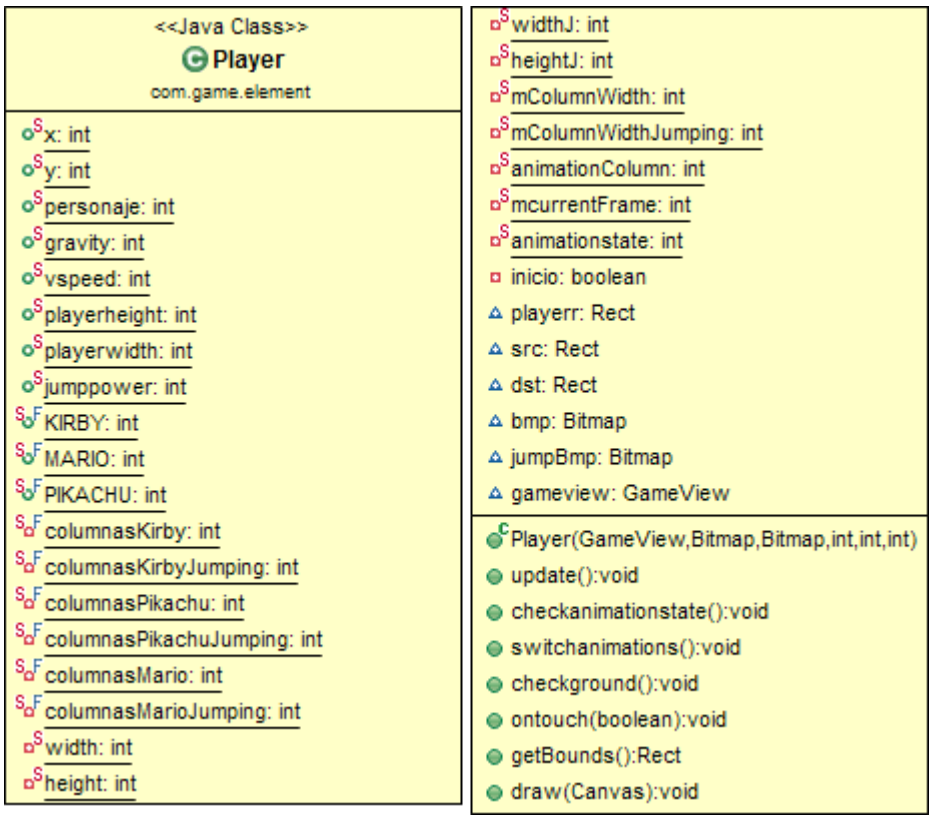


Figura IV.III.V.I Diagrama de clase *Player*

- *Enemy* (figura IV.III.V.II): todos los enemigos que cada personaje va a tener que enfrentar durante su partida. Actualmente hay tres: *Gordo* para *Kirby*, *Planta Piraña* para *Mario* y *Raichu* para *Pikachu*.

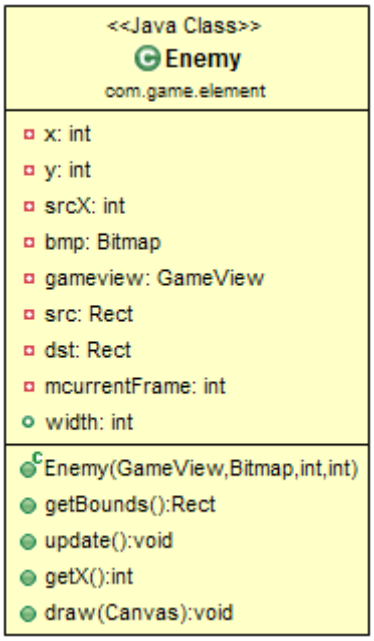


Figura IV.III.V.II Diagrama de clase *Enemy*

- *Coin* (figura IV.III.V.III): objeto moneda que se podrá recolectar durante todo el juego.

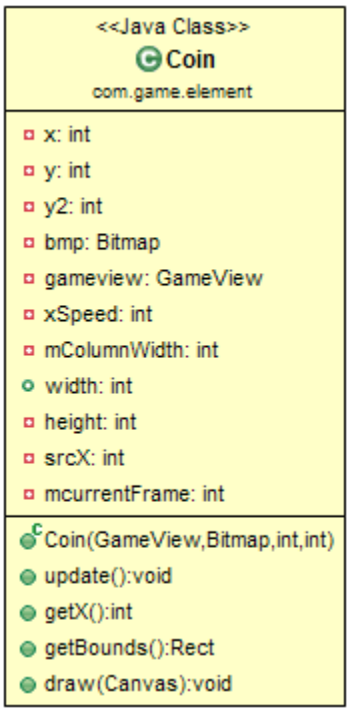


Figura IV.III.V.III Diagrama de clase *Coin*

- *Cloud* (figura IV.III.V.IV): elemento pasivo que proporciona al juego sensación de profundidad y dinamismo.

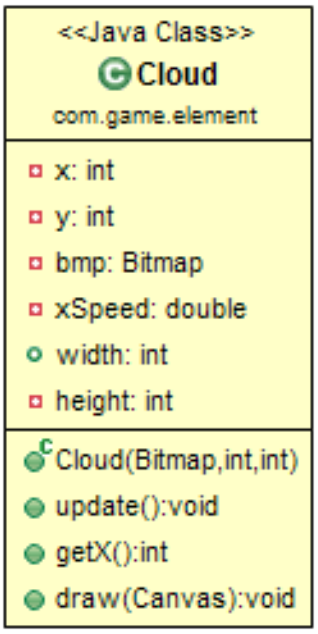


Figura IV.III.V.IV Diagrama de clase *Cloud*

- *PowerUP* (figura IV.III.V.V): mejoras temporales para el personaje.

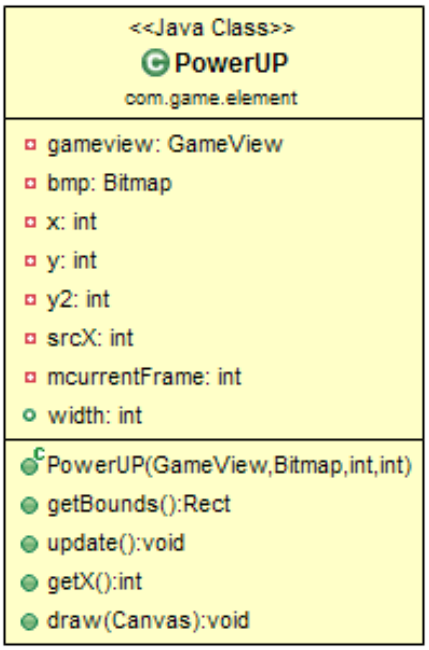


Figura IV.III.V.V Diagrama de clase *PowerUP*

- *Ground* (figura IV.III.V.VI): objeto suelo que rellenará la parte inferior de la pantalla.

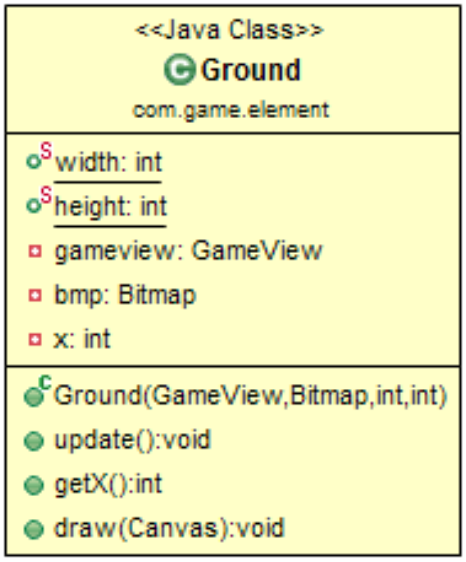


Figura IV.III.V.VI Diagrama de clase *Ground*

- *Button* (figura IV.III.V.VII): objeto botón, el cual aparece cuando muere el personaje cuyo uso actualmente es reiniciar el juego.

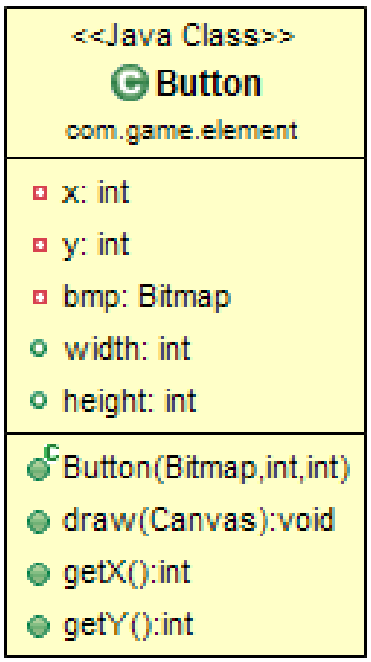


Figura IV.III.V.VII Diagrama de clase *Button*

IV.III.VI Herramientas

Por la necesidad de utilizar ciertas funciones comunes a todo el código, se decidió unificarlas todas en una clase estática llamada *Tools* (figura IV.III.VI.I) desde la que poder ejecutarlas sin necesidad de instanciar la clase.

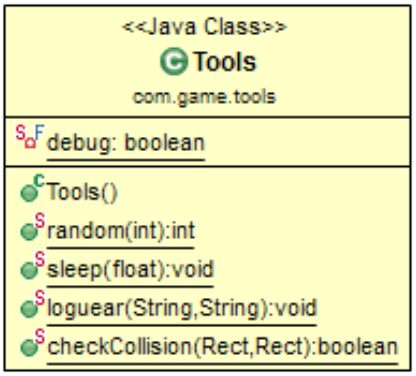


Figura IV.III.VI.I Diagrama de clase *Tools*

Esta clase ha resultado ser especialmente útil para la realización de todas las pruebas.

IV.IV Diagramas de secuencia

En esta sección se expondrá el diagrama de secuencia referente a la interacción resultante entre las clases de la aplicación durante la ejecución de un ciclo del bucle de *GameLoopThread*, es decir, durante el proceso de creación de un frame.

Se ha escogido mostrar únicamente este diagrama pues consideramos que es el núcleo principal de las interacciones entre clases de esta aplicación y además muestra ejemplos de la gran parte de los tipos de interacciones que se dan en ella.

Para facilitar la explicación de la iteración del bucle y poder insertar el diagrama de tal forma que quedase lo más visual posible, lo hemos dividido en dos partes. La primera engloba la primera mitad del bucle, donde se realizan las labores de reciclado de memoria, mediante la eliminación de elementos y la creación de nuevos elementos. La segunda mitad contiene la parte del bucle en la cual se actualizan y dibujan en pantalla todos los elementos.

Consideraremos en las explicaciones a realizar que las clases *Ground*, *Cloud*, *Enemy*, *Coin* y *PowerUP* representadas en los diagramas no son siempre la misma clase, sino que se referirán a todas las instancias que existan en un determinado momento del juego de cada clase y, además, a las que están por crear. Asimismo, se considera que en la iteración del bucle a representar el juego está en ejecución y el personaje no va a morir.

IV.IV.I Iteración de *GameLoopThread*. Primera parte

Esta primera mitad comienza con la llamada al método *onDraw()* de *GameView* por parte de *GameLoopThread*.

Una vez dentro del método *onDraw()*, *GameView* se encarga de ejecutar la llamada a *update()* método encargado de realizar las labores de reciclado de memoria, gestión del suelo y creación de nuevos elementos.

El proceso de reciclado de memoria es similar para todos los elementos, excepto para el suelo. El método *update()* comienza llamando a la función *deleteGround()* la cual se encarga de recorrer un *ArrayList* que almacena todos los objetos *Ground* existentes y comprueba uno a uno si su coordenada x es menor que cero, mediante la llamada a *getX()*. Si este fuera el caso, significa que ese objeto debería ser dibujado a la izquierda de la pantalla, por lo que no podría ser visualizado y el objeto tan solo ocuparía espacio en memoria. Por este motivo, cuando esto sucede, esos objetos *Ground* son eliminados. Si tan solo se eliminasen objetos *Ground* llegaría un momento en el que todo el suelo desaparecería, por lo tanto es necesario que se creen tantos objetos *Ground* como se hayan eliminado. Esto se representa en el diagrama con la llamada a *Ground(GameView, Bitmap, int, int)*, la cual se ejecutará un número determinado de veces dependiendo de cuantos objetos se hayan borrado, generando de este modo más suelo por la derecha.

Para el caso de los objetos tipo *Cloud*, *Enemy*, *Coin* y *PowerUP*, el proceso es similar al de *Ground*, salvo la parte de generar nuevos objetos, la cual se expondrá a continuación.

Una vez se ha terminado la fase de reciclado de memoria, se procede a incrementar, mediante la llamada a *updateTimers()*, los contadores que controlan la aparición de nuevos objetos tipo *Cloud*, *Enemy*, *Coin* y *PowerUP*.

A partir de este momento, supondremos que en la iteración en la que se encuentra el bucle se generarán elementos de todos los tipos.

De este modo y una vez actualizados los contadores, el programa llamará a la función *generateElements()*, la cual comprobará si es momento de generar nuevos elementos de forma pseudoaleatoria. Considerando que esto es correcto, el programa generará, por orden, un nuevo bonificador mediante la llamada *PoweUP(GameView, Bitmap, int, int)*, cinco nuevas monedas mediante la llamada a *Coin(GameView, Bitmap,*

int, int), un nuevo enemigo mediante *Enemy(GameView, Bitmap, int, int)* y una o dos nuevas nubes mediante *Cloud(Bitmap, int, int).*

Una vez llegado a este punto la función *update()* termina su cometido y el proceso continua desde la función *onDraw()* de *GameView*.

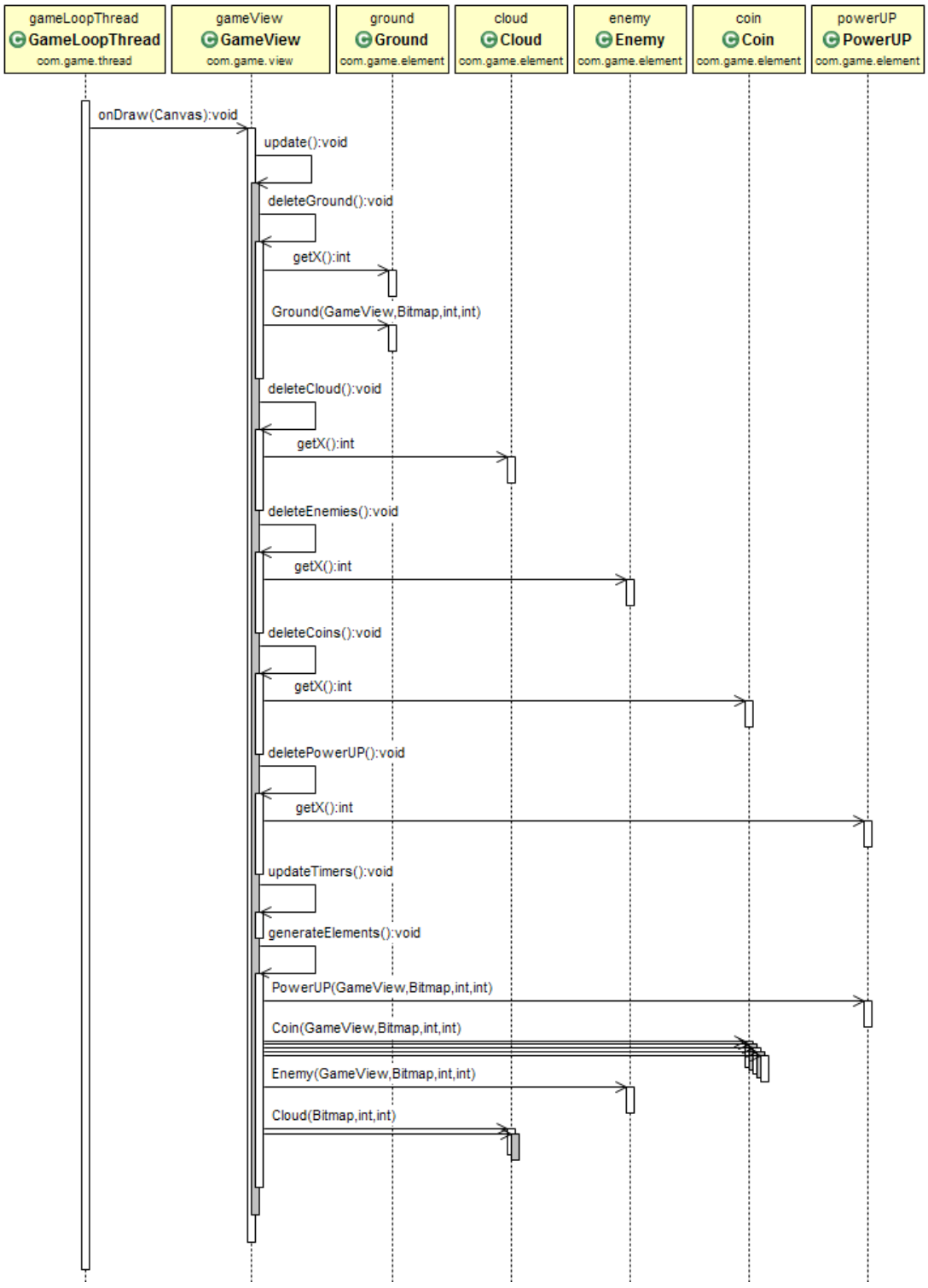


Figura IV.IV.I.I Diagrama de secuencia iteración de *GameLoopThread* 1

IV.IV.II Iteración de *GameLoopThread*. Segunda parte

Una vez ejecutado el método *update() GameView* llama al método *drawEnvitonment()*. Este método será el encargado de realizar las llamadas a cada uno de los métodos draw de los elementos, los cuales actualizarán el estado de cada uno de ellos y lo dibujarán por pantalla.

El proceso comienza con la llamada al método *getBounds()* del objeto *Player*. Este método devuelve un objeto de tipo *Rect* que define los 4 puntos de coordenadas que encierran al sprite del jugador en su posición actual en la pantalla y que será utilizado posteriormente para realizar las comprobaciones de colisiones con los correspondientes elementos.

Una vez se tiene el *Rect* del jugador, se procede a dibujar los elementos que no requieren de comprobación de colisión, esto es, el suelo y las nubes. Para ello se ejecuta una llamada a un bucle tipo *for* que recorre el *ArrayList* que almacena cada uno de estos tipos de elementos y procede a realizar la llamada al método *draw(Canvas)* de cada uno de los elementos del *ArrayList*.

A continuación se realizan las llamadas a los métodos draw de aquellos elementos que sí requieren comprobación de colisiones, esto es, *Enemy*, *Coin* y *PowerUP*.

Este proceso de dibujo y comprobación comienza con los objetos de tipo *Enemy*, ya que, en el caso de existir una colisión, no sería necesario seguir con el método puesto que el jugador habría muerto. En primer lugar, tras la creación e inicialización del bucle *for* que recorrerá el *ArrayList* de *Enemy*, se realiza la llamada al método *draw(Canvas)* de cada uno de los elementos del *ArrayList* que, como ya hemos dicho anteriormente, actualizará el estado del objeto que le llame y lo dibujará por pantalla. Posteriormente se llamará al método *getBounds()*, obteniéndose así el *Rect* del enemigo correspondiente. Antes de finalizar la iteración del bucle *for*, se realiza la comprobación mediante la llamada a *checkCollision(Rect, Rect)* de la clase *Tools*, que devolverá *true* en caso de colisión o *false* si no la ha habido.

Este proceso se ejecutará por cada enemigo del *ArrayList* y en el caso de existir colisión se procederá a detener el método ya que el jugador habrá muerto. Suponiendo que esto no haya sido así y que el jugador siga vivo, este proceso de dibujo y

comprobación se repetirá para el *ArrayList* de *Coin* y el de *PowerUP*, comprobando las colisiones en cada caso y actuando según corresponda.

Para el caso de las monedas, se incrementará el contador de monedas recogidas por cada una de ellas y aumentará la puntuación de la partida. Para el caso de los bonificadores se activará el mismo para el jugador, proporcionándole un tiempo de mejores habilidades.

Una vez hecho esto se volverá al método *onDraw(Canvas)* de *GameView*, donde antes de finalizar la ejecución de la iteración del bucle se llamará al método *draw(Canvas)* de *Player*. Al igual que en los casos anteriores este método se encargará de actualizar el estado del personaje (frame de su animación a mostrar dependiendo de cada situación) y de dibujar al mismo por pantalla.

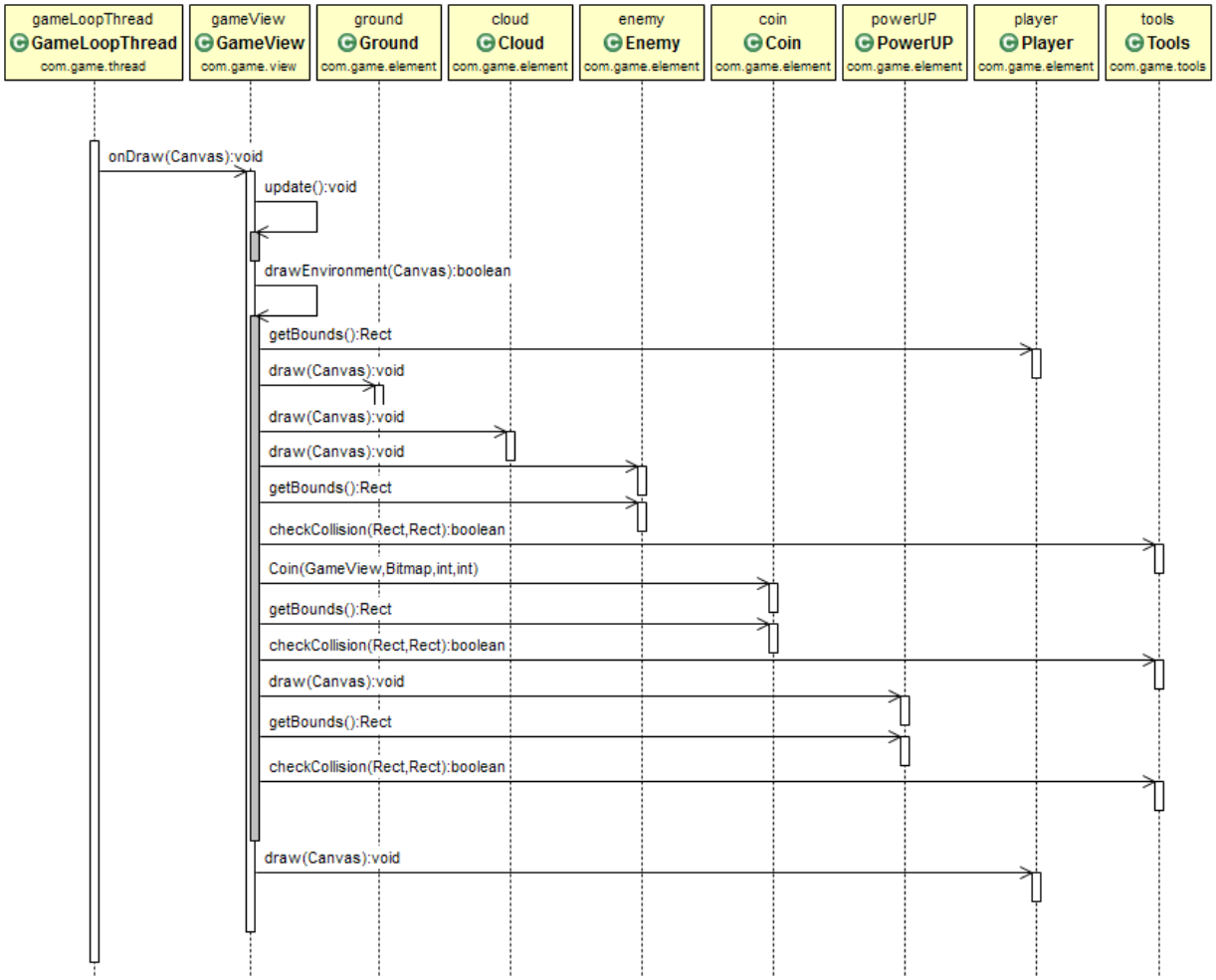


Figura IV.IV.I.I Diagrama de secuencia iteración de *GameLoopThread* 2

Capítulo V: Pruebas de la aplicación

El objetivo de este capítulo es presentar las pruebas realizadas a lo largo de todo el desarrollo de la aplicación.

Las pruebas realizadas se dividen en pruebas unitarias, de integración y de sistema y el objetivo de todas ellas es asegurar el correcto funcionamiento de la aplicación. Estas pruebas constan de una serie de investigaciones empíricas y técnicas, cuyo objetivo es proveer de información objetiva acerca de la calidad de un producto a la parte interesada o stakeholder.

V.I Pruebas unitarias

Las pruebas unitarias son aquellas destinadas a probar la funcionalidad de ciertos módulos de código. Suelen realizarse en paralelo a la fase de implementación de tal forma que cada nuevo núcleo de código puede probarse y asegurar que su funcionamiento de forma individual es correcto. De este modo, las pruebas unitarias, en nuestro caso, fueron realizadas según se implementaban nuevos métodos, tal y como indica nuestra metodología, y al final de cada semana, con el fin de reafirmar la corrección de los métodos previamente al inicio de una nueva iteración.

En nuestro caso se ha buscado comprobar la funcionalidad de ciertos métodos relevantes para el conjunto de la aplicación. A estos métodos se les aplicaron pruebas de tipo caja negra y de tipo caja blanca.

Las pruebas de caja negra se basan en comprobar la entrada y la salida de un método y asegurar que se recibe lo esperado dependiendo de la entrada sin reparar en nada del proceso interno que sigan los datos.

Las pruebas de caja blanca, por el contrario, se basan en, dada una entrada observar su evolución a lo largo de una función comprobando que se recorren todos los caminos posibles y que se obtiene la salida esperada.

V.I.I Pruebas de caja negra

Dada la naturaleza de la aplicación, es necesario destacar que el número de métodos con entrada y salida desarrollados es muy limitado. Por este motivo el número de pruebas de caja negra fue bastante bajo, ya que este tipo de pruebas no eran adecuadas para observar el estado de la aplicación.

En algunos métodos, pese a no existir salida como tal, sí se produce un efecto sensible como consecuencia de la ejecución del método y, por lo tanto, se comparó este efecto a la salida del mismo.

De este modo los únicos métodos a los que se les aplicó pruebas de caja negra fueron a los métodos de *EndlessTributeService*, ya que su función radica o bien en reproducir un sonido o bien en detenerlo. Los métodos por tanto a los que se les aplicaron las pruebas de caja negra son los siguientes:

- *startSong(int)*: método que carga una canción definida por un valor numérico e inicia su reproducción.

- *stopSong()*: método que detiene la reproducción de una canción.
- *pauseSong()*: método que pausa la reproducción de una canción.
- *resumeSong()*: método que reanuda la reproducción de una canción.
- *playSound(int)*: método que reproduce un sonido FX.

En la tabla V.I.I.I se ofrece un resumen de las pruebas ejecutadas sobre estos métodos, explicando los supuestos en cada caso, la entrada, si la hay, y la salida. Las pruebas que dependen de una entrada se han ejecutado en repetidas ocasiones con entradas distintas, aunque por sencillez la tabla solo muestra un caso. Para todos los casos se supone que el reproductor de música se encuentra inicializado.

Método	Supuestos	Entrada	Salida esperada	Salida obtenida	Estado
startSong(int)	-	R.raw.ssbbintro	Reproducción de R.raw.ssbbintro	Reproducción de R.raw.ssbbintro	**Correcto**
stopSong()	Canción en reproducción	-	Canción detenida	Canción detenida	**Correcto**
pauseSong()	Canción en reproducción	-	Canción pausada	Canción pausada	**Correcto**
resumeSong()	Canción en pausa	-	Canción en reproducción	Canción en reproducción	**Correcto**
playSound(int)	-	SOUND_kirbyhi1	Reproducción de SOUND_kirbyhi1	Reproducción de SOUND_kirbyhi1	**Correcto**

Tabla V.I.I.I Pruebas unitarias de caja negra

V.I.II Pruebas de caja blanca

Dada la naturaleza de este tipo de pruebas, en las cuales se requiere analizar completamente todos los posibles caminos que puede seguir la ejecución normal de todos los métodos, resulta poco práctico mostrar los resultados y siquiera el proceso en un documento de este tipo.

Por este motivo tan solo se mostrará el proceso de análisis para el método *checkanimationstate()* de la clase *Player* pudiendo ser extrapolado a la mayoría de los demás métodos de la aplicación.

Para realizar los análisis de las pruebas de caja blanca, se ha seguido el *criterio de cobertura de caminos*[7]. Este método se caracteriza por describir casos de prueba suficientes como para comprobar todos y cada uno de los caminos desde el inicio del método hasta su fin. En este contexto un camino una secuencia ininterrumpida de instrucciones.

[7] http://www.lsi.us.es/docencia/get.php?id=361

El análisis comienza con el estudio del método en cuestión. La siguiente imagen muestra el código del método.

```
public void checkanimationstate() {
    // Inicio salto
    if (vspeed < 0) {
        animationstate = 2;
        if(inicio){
            mcurrentFrame = 0;
            inicio = false;
        }
    // Caida
    } else if (vspeed > 0) {
        animationstate = 1;
    // En el suelo
    } else {
        animationstate = 0;
        inicio = true;
    }
}
```

Figura V.I.II.I Método *checkanimationstate()*

Como se puede observar, en la función existen tres caminos posibles y dentro del primero de ellos, una parte opcional. De aquí se puede extraer el diagrama de flujo asociado a este método, así como el grafo extraído a partir del de flujo, que se muestran a continuación.

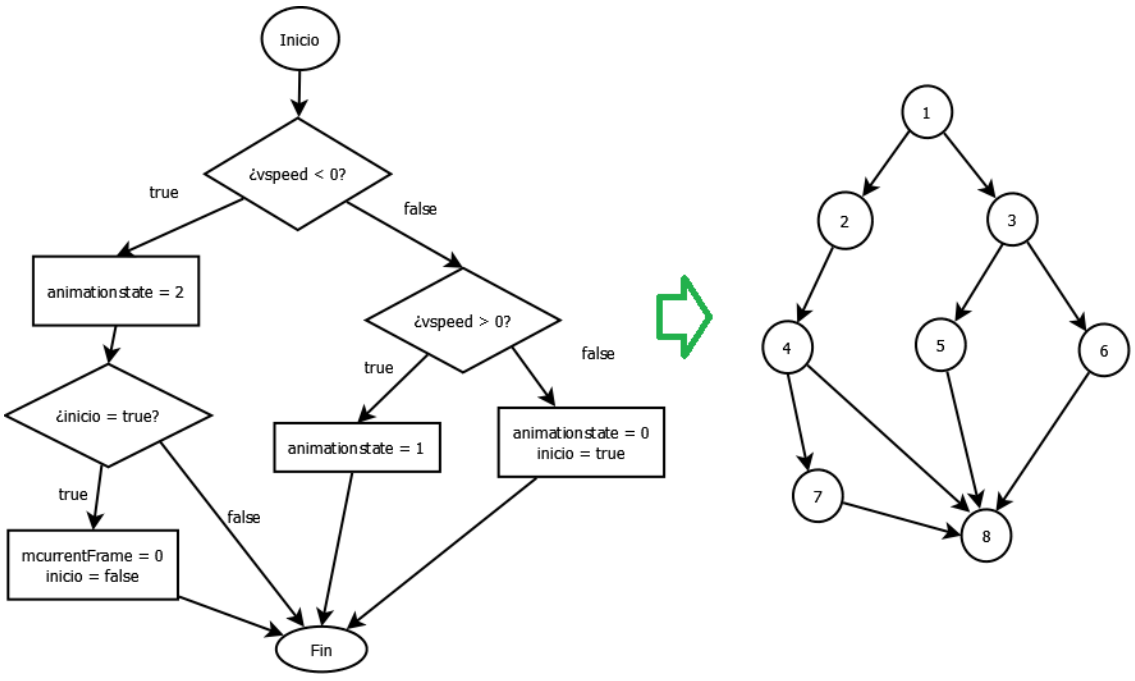


Figura V.I.II.II Diagrama de flujo y grafo de *chechanimationstate()*

Una vez hecho esto es necesario calcular la complejidad ciclomática, que nos determinará el número de caminos independientes de la función.

Existen tres formas de calcular la complejidad ciclomática y son estas:

1. Número de regiones del grafo.
2. Número de aristas menos número de nodos más dos.
3. Nodos predicado más uno.

De este modo, la complejidad ciclomática para nuestro caso es 4.

1. Número de regiones del grafo = 4 (figura V.I.II.III).
2. Aristas - Nodos + 2 = 10 - 8 + 2 = 4.
3. Nodos predicado + 1 = 3 +1 = 4.

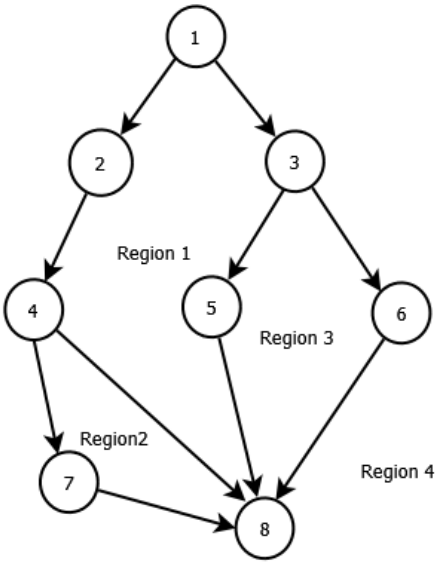


Figura V.I.II.III Regiones del grafo de *checkanimationstate()*

Una vez hecho esto es necesario localizar los caminos independientes siguiendo unos criterios concretos. Estos criterios se describen a continuación.

1. Elegir el camino que atraviese el máximo número de decisiones y que no se trate de un error en la ejecución.
2. Elegir un segundo camino alterando la primera decisión del primer camino e intentando mantener las demás igual.
3. Elegir un tercer camino manteniendo la primera decisión del primer camino, alterando la segunda y manteniendo las demás.
4. Continuar con el proceso hasta encontrar todos los caminos.

De este modo, una vez llegados a este punto somos capaces de identificar cada uno de los caminos independientes donde el código repetido es mínimo. Eso a su vez

nos permite diseñar un caso de prueba concreto para cada camino y así poder realizar nuestras pruebas de caja blanca.

La siguiente tabla muestra los resultados de la búsqueda de caminos, así como las pruebas diseñadas para cada uno, el resultado esperado y obtenido y el estado de la prueba.

Camino	Precondición	Resultado esperado	Resultado obtenido	Estado
1-2-4-7-8	vspeed < 0 inicio = true	animationstate = 2 mcurrentFrame = 0 inicio = false	animationstate = 2 mcurrentFrame = 0 inicio = false	**Correcto**
1-3-5-8	vspeed > 0	animationstate = 1	animationstate = 1	**Correcto**
1-2-4-8	vspeed < 0 inicio = false	animationstate = 2	animationstate = 2	**Correcto**
1-3-6-8	vspeed = 0	animationstate = 2 inicio = true	animationstate = 2 inicio = true	**Correcto**

Tabla V.I.II.IV Pruebas unitarias de caja blanca

V.II Pruebas de integración

Se considera pruebas de integración a la consecución de varias pruebas unitarias juntas, es decir, a la realización de una prueba en la que se valoran todos los métodos que componen un proceso en conjunto, desde que se inicia el proceso hasta que termina.

Dada la extensión de este tipo de pruebas, consideramos que no es correcto mostrarlas en un documento de esta índole y por lo tanto no serán incluidas en el mismo. Además, dado que ya se han mostrado los métodos seguidos para realizar las pruebas unitarias se pueden extrapolar los mismos para las de integración.

V.III Pruebas de sistema

Estas pruebas se encuentran divididas en dos etapas. La primera de ellas buscaba comprobar el correcto funcionamiento de los dos subsistemas existentes en la aplicación y, posteriormente, el de la aplicación tras la integración de ambos subsistemas. Esta primera etapa se realizó al final de la fase de desarrollo y al comienzo de la fase de pruebas. La segunda buscaba comprobar la satisfacción del usuario y la cumplimentación de objetivos. Esta segunda etapa se realizó al final de la fase de pruebas, cuando la aplicación se encontraba en una versión estable, y lista para ser distribuida a los usuarios del entorno de pruebas.

V.III.I Pruebas de subsistemas y sistema completo

El proceso de desarrollo de la aplicación se segmentó en dos fragmentos independientes en un primer momento. Esto permitía ir realizando pruebas más rápidas y fácilmente sin tener que tratar con una aplicación demasiado grande donde había muchos puntos de posibles errores.

Las dos partes en las que se dividió la aplicación fueron las siguientes:

- Subsistema de *Android*: consta de todas las actividades que tienen que ver con el entorno de menús del juego y del servicio que da soporte a toda la aplicación. Este subsistema se enfoca a la funcionalidad de *Android* más que al propio juego.
- Subsistema del juego: consta de la actividad *GameActivity*, de la vista *GameView*, del hilo *GameLoopThread* y de todos los elementos que aparecen en el juego. Este subsistema se enfoca a la funcionalidad del juego en sí.

Para realizar las pruebas en ambos subsistemas por separado se esperó hasta que ambos estuviesen terminados en una versión estable. Una vez esta fase fue alcanzada, se procedió a probar ambas partes por separado en entorno de pruebas utilizando la herramienta *LogCat* para el depurado de ambas partes. El objetivo de esta fase de pruebas era el de comprobar la correcta funcionalidad de la aplicación por separado, previamente a realizar la integración de ambos subsistemas y a la realización de las pruebas centradas en el usuario.

La realización de estas pruebas se basó en gran medida en la navegación por ambas partes de la aplicación comprobando mediante *LogCat* y mensajes de impresión

de trazas que toda la ejecución era correcta. Si algún error era detectado por el equipo de pruebas, se comunicaba al desarrollador, que procedía a corregirlo y a generar una nueva versión estable con el error ya corregido. Tras una serie de iteraciones de este tipo de pruebas se dio por finalizada la fase tras no encontrar ningún defecto más en el código, por lo que se procedió a la integración de ambas partes. Este proceso fue relativamente sencillo, ya que se sabía perfectamente qué aspectos de ambos subsistemas debían ser modificados, lo que llevado a la práctica resultó en una integración rápida y sencilla de ambas partes. Una vez estuvieron ambas partes integradas, se volvió a ejecutar el proceso de pruebas en entorno cerrado. El equipo de pruebas realizó la navegación por la aplicación en repetidas ocasiones, esta vez sin encontrar ningún tipo de defecto, por lo que se consideró que la aplicación estaba lista para ser probada por los usuarios de pruebas.

Para más información al respecto de las pruebas realizadas al sistema completo, consultar Apéndice II.

V.III.II Pruebas enfocadas a usuario y cumplimentación de objetivos.

Dado que durante todo el proyecto se han utilizado metodologías ágiles de desarrollo, en las pruebas destinadas al usuario y a la cumplimentación de objetivos también se tendrá al usuario objetivo, es decir, el potencial cliente, como parte de las pruebas. De este modo este apartado se centrará en conocer la opinión del público respecto a la aplicación y a su valoración sobre si se han alcanzado o no los objetivos.

Una vez se hubo terminado con la fase de pruebas en entorno cerrado se procedió a distribuir la aplicación entre usuarios cercanos que accedieron a formar parte de las pruebas. A estos usuarios se les instaló en su dispositivo móvil la aplicación o bien se les prestó un móvil con la aplicación instalada, se les entregó el manual de usuario y se les dejó probar el juego sin informarles sobre nada. Tras un rato jugando se les entregó una pequeña encuesta para conocer su opinión acerca de la aplicación.

A continuación se muestran los resultados de esta encuesta. Cada una de las tres tablas muestra cada pregunta realizada en la encuesta y la respuesta que mayor porcentaje de votos obtuvo, junto al porcentaje.

Pregunta	Opción más votada	Porcentaje de votos
¿Considera la dinámica del juego complicada?	1, completamente en desacuerdo	85%
Valore la dificultad de Kirby (1 ninguna, 5 imposible)	3	40%
Valore la dificultad de Mario (1 ninguna, 5 imposible)	1, nada	100%
Valore la dificultad de Pikachu (1 ninguna, 5 imposible)	1, nada	80%
¿Considera el incremento de la velocidad excesivo?	1, completamente en desacuerdo	90%
¿Considera que el juego responde bien a los toques en la pantalla?	5, completamente de acuerdo	100%
¿Considera que el juego responde bien a los movimientos del sensor?	4	70%

Tabla V.III.II.I Resultados sobre la jugabilidad

Pregunta	Opción más votada	Porcentaje de votos
¿Considera que son sencillos e intuitivos?	5, completamente de acuerdo	100%
¿Entiende fácilmente lo que hacen las opciones del juego?	5, completamente de acuerdo	60%
¿Cumplen su función las opciones del juego?	5, completamente de acuerdo	100%

Tabla V.III.II.II Resultados sobre los menús

Pregunta	Opción más votada	Porcentaje de votos
¿Le gustaría que apareciesen más personajes?	Sí	100%
En caso de que el juego se comercializase, ¿pagaría por él?	Sí	80%
¿Le resultó útil el manual de usuario?	Sí	55%

Tabla V.III.II.III Resultados sobre el juego

Como se puede observar de estos datos, la aplicación, incluso en una versión de prueba de concepto, ha tenido una gran acogida entre los usuarios que la han probado. Hemos de destacar que este público se encuentra entre los nueve y los cincuenta y siete años.

Capítulo VI: Conclusiones y líneas futuras

En este capítulo se presentan las conclusiones extraídas después de realizar este Trabajo Fin de Grado, así como las líneas futuras de desarrollo de la aplicación, así como las posibles mejoras que se pudiesen implementar.

VI.I Conclusiones

A lo largo de esta memoria se ha descrito el proceso de desarrollo completo de un videojuego para *Android*.

Debido a la metodología elegida para la realización de este sistema, la aplicación resultante es estable, ya que se han ido realizando pruebas tras la finalización de cada hito.

Tras el estudio de la facilidad de manejo de videojuegos de este tipo ya existentes en el mercado, se ha podido realizar una aplicación de uso intuitivo y con un menú de fácil manejo. De esta forma, el grado de satisfacción de los usuarios se espera que sea, como ya se ha comprobado anteriormente, mayor, por lo que se alcanzará un público objetivo más alto.

Una de las ventajas que presenta este videojuego, es que se puede interactuar tanto pulsando en la pantalla como moviendo el terminal móvil. Así, los usuarios pueden jugar de la forma en la que se sientan más cómodos, mejorando su experiencia.

Dicho esto, volveremos a repasar los objetivos de este proyecto mencionando en cada caso el estado actual de los mismos, explicando si se han completado o no y los motivos de este hecho:

- **Realizar un videojuego para dispositivos móviles *Android*, teniendo por objetivo un público con un margen de edad lo más amplio posible y con un resultado atractivo para todo ese público:** Este es el objetivo principal de este trabajo, el cual se ha cumplido de todo modo, tal y como ha quedado demostrado gracias a las encuestas realizadas a los usuarios y descritas en el capítulo V sección III.II. Por este motivo se considera este objetivo completado.
- **Exportar los modelos exitosos del mercado tradicional del videojuego a un nuevo mercado centrado en los dispositivos móviles:** Este objetivo complementa al primero, ya que para realizar nuestro juego ha sido necesario exportar videojuegos exitosos del mercado tradicional al nuevo mercado de los dispositivos móviles. Por este motivo se considera este objetivo completado.
- **Aprovechar un sector del mercado no explotado para crear e introducir una necesidad, que pueda ser aprovechada**

posteriormente comercializando nuestro producto: Este objetivo se deriva de los dos anteriores. Una vez hemos creado un producto atractivo para el público, exportando los modelos exitosos del mercado tradicional, aprovecharemos el hueco no explotado por las grandes desarrolladoras existente en este mercado para introducir la necesidad de adquirir nuestro producto y que este pueda ser comercializado. Además, tal y como se ha demostrado en el capítulo V sección III.II, existe público dispuesto a adquirir este producto una vez se comercializase. Por estos motivos consideramos este objetivo completado.

Durante la realización de este trabajo, se han puesto en práctica gran parte de los conocimientos, por no decir todos, adquiridos durante estos cuatro años de carrera. Sin embargo, han tomado una especial importancia aquellas asignaturas en las cuales se han estudiado con más detenimiento los procesos de desarrollo de software, optimización de recursos, programación, tanto en *Java* como enfocada a *Android* y diseño de interfaces de usuario. Además de lo aportado por la universidad, también ha sido necesario recurrir al estudio individual en mayor profundidad de *Android*, experiencia que se valora muy positivamente, dada la importancia que tiene este sistema operativo hoy en día.

Me gustaría destacar concretamente el hecho de haber desarrollado una aplicación con interfaz gráfica por mí mismo y desde cero. Este hecho no había sido realizado anteriormente en la carrera en ninguna asignatura, ya sea porque se nos suministraba la interfaz gráfica ya programada o porque a la mayoría de los programas que creábamos se accedía por línea de comandos. Por lo tanto, me siento muy orgulloso de haber conseguido realizar esta aplicación a la cual, aun tratándose en esencia de una prueba de concepto, pretendo dar continuidad en el fututo siguiendo las líneas futuras que se expondrán a continuación.

VI.II Líneas futuras

En esta sección se expondrán las ideas que se llevarán a cabo en un futuro para mejorar la aplicación y otras que serán más difíciles de conseguir.

VI.II.I Mejora de los gráficos

Como ya se explicó en capítulos anteriores, los gráficos del juego se han extraído de otros videojuegos. Pese a que ello no supone un problema de cara a la funcionalidad de la aplicación, sí que convendría de cara a un posible lanzamiento comercial unificar todos los gráficos al mismo tipo de diseño. No obstante, este detalle provocaría que el juego dejase de tener ese carácter de homenaje que se le ha pretendido imprimir.

VI.II.II Refactorización del código

Siguiendo las líneas de la metodología aplicada al proyecto, *Extreme Programming*, previamente a realizar cualquier tipo de mejora o implementación extra será necesario realizar un análisis de refactorización del código. Actualmente, pese a ser una versión completamente estable para la mayoría de las versiones de *Android*, al tratarse de una prueba de concepto no se han tenido en cuenta las diferentes capacidades de los personajes con el fin de que a cada personaje le correspondiese una clase que heredase de la clase Player, sino que las características de cada uno de los tres personajes habitan dentro de ella. Por este motivo, será necesario realizar una refactorización, al menos de cara a este punto.

VI.II.III Ampliación de personajes y especialización de cada uno

Tal y como se explicó en capítulos anteriores, como prueba de concepto se han desarrollado tan solo tres personajes, pero se pretende la inclusión de muchos otros. Además, por el contrario de cómo funciona en este momento la aplicación en la que los tres personajes realizan las mismas acciones, se pretende especializar a cada uno de los personajes del juego de tal forma que sus posibles acciones sean coherentes con las acciones del personaje en su juego original.

VI.II.IV Creación y utilización de una banda sonora propia o adaptada

Como ya se explicó en el Capítulo IV la banda sonora utilizada en este momento en el videojuego no es propia, sino que ha sido extraída de otro videojuego. De cara a una posible comercialización del mismo se debería obtener una banda sonora propia o al

menos conseguir una versión adaptada al dispositivo. Esta línea de desarrollo puede resultar complicada sin tener los contactos adecuados.

VI.II.V Obtención de las licencias de uso asociadas a cada personaje

Como prueba de concepto y sin ánimo de lucro, se han utilizado personajes propiedad de grandes compañías sin su permiso. Uno de los principales objetivos a conseguir de cara al futuro pasa por presentar la aplicación a las correspondientes compañías en pos de obtener las licencias oportunas para poder distribuir legalmente la aplicación o, en el caso de ofrecerse la oportunidad, vender la idea a la compañía que quiera adquirirla.

Bibliografía

[1] EXTREME PROGRAMMING [en línea]

<http://www.extremeprogramming.org>

[2] DAVIDVALVERDE, blog [en línea]

<http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp>

[3] LABIBLIADELPROGRAMADOR, Estructura Android [en línea]

< http://labibliadelprogramador.blogspot.com.es/2012/09/estructura-de-android.html>

[4] THE IMPOSSIBLE GAME, página del juego [en línea]

<http://www.flukedude.com/theimpossiblegame>

[5] IMANGI STUDIOS, Temple Run [en línea]

<http://www.imangistudios.com>

[6] HALFBRICK STUDIOS, Jetpack Joyride [en línea]

<http://halfbrick.com/our-games/jetpack-joyride>

[7] ADMINISTRACIÓN ELECTRÓNICA, Métrica v3 [en línea]

<http://administracionelectronica.gob.es/?_nfpb=true&_pageLabel=P800292251293651550991&langPae=es&detalleLista=PAE_000000432>

[8] MOTOR DE DESARROLLO [en línea]

<http://en.wikipedia.org/wiki/Game_engine>

Apéndices

Apéndice I: Presupuesto del proyecto

Una vez conocido el esfuerzo asociado a la realización de este Trabajo Fin de Grado se procederá a detallar los costes en los que se ha incurrido.

En primer lugar se hará un balance con los gastos asociados al esfuerzo humano en base a las horas aportadas al proyecto por cada individuo. Se miden horas y no días por considerar que no se ha trabajado días completos por motivos obvios. Estas horas serán desglosadas acorde al tipo de tarea desempeñada dentro del proyecto. De este modo hemos convenido dividir las tareas en los siguientes tipos:

- Conceptualización y Diseño (CyD): a un coste de 20 €/hora
- Codificación: a un coste de 12 €/hora
- Pruebas: a un coste de 15 €/hora
- Redacción: a un coste de 10 €/hora
- Asesoramiento: a un coste de 10 €/hora
- Revisión: a un coste de 10 €/hora

Además, se contemplará la categoría de cada individuo, que será diferenciada dentro de tres tipos. De este modo, a cada categoría le corresponde un multiplicador sobre el coste descrito tal y como se indica a continuación:

- Ingeniero junior: multiplicador 1.0
- Ingeniero advance: multiplicador 1.5
- Ingeniero senior: multiplicador 2.0

Como personal que ha colaborado en el proyecto se ha incluido a Diego Trompeta Urretavizcaya, como desarrollador de la aplicación, a Jorge Ruiz Magaña, como tutor del Trabajo, el cual aportó horas en forma de reuniones, asesoramiento y revisiones, a Santiago García Delgado, el cual aportó horas en forma de asesoramiento y, por último, Carolina García Vázquez, quien aportó horas en forma de asesoramiento y revisión.

A continuación se muestra la tabla acerca de los gastos asociados al esfuerzo humano.

Nombre	Categoría	Tarea	Coste/hora	Duración (horas)	Coste (Euros)
Diego Trompeta Urretavizcaya	Ingeniero Junior	CyD	20	80	1600
		Codificación	12	160	1920
		Pruebas	15	40	600
		Redacción	10	120	1200
Jorge Ruiz Magaña	Ingeniero Senior	Asesoramiento	20	10	200
		Revisión	20	20	400
Santiago García Delgado	Ingeniero Advance	Asesoramiento	15	5	75
Carolina García Vázquez	Ingeniero Senior	Asesoramiento	20	7	140
		Revisión	20	10	200
			Total	**472**	**6335**

Tabla Apéndice I.I Costes asociados al esfuerzo humano

Además de estos gastos relacionados con el esfuerzo humano es necesario calcular los gastos asociados a los equipos utilizados durante la realización de este proyecto. Dado que la duración del proyecto ha sido de cinco meses y que los equipos asociados al mismo no han sido dedicados en su totalidad a la realización del proyecto, se harán los cálculos teniendo en cuenta el factor de utilización asociado a los mismos.

Como periodo de vida útil de los equipos, de cara a las amortizaciones, se considera que los equipos de sobremesa tienen una vida útil de 48 meses, los portátiles de 36 meses y los dispositivos móviles de 24 meses.

El coste, por tanto, asociado a cada equipo responderá a la siguiente fórmula:

$$\frac{\text{Coste del producto (€)} \cdot \text{Tiempo de uso (Meses)} \cdot \text{Factor de utilización (0.0 – 1.0)}}{\text{Periodo de vida útil (Meses)}}$$

Figura Apéndice I.II Ecuación de cálculo de amortización de equipos

A continuación se muestra la tabla acerca de los gastos asociados a las amortizaciones de equipos informáticos.

Descripción	Coste (€)	Tiempo de uso (meses)	Factor de utilización (0.0-1.0)	Periodo de vida útil (meses)	Coste imputable (€)
Ordenador de sobremesa *Intel* i7-3770 3.40 GHz, 16GB RAM	2000	4	0.8	48	133.33
Ordenador portátil Compaq Pavilion g6t-2000, *Intel* i3-2350M 2.30 GHz, 4GB RAM	480	2	0.2	36	5.33
Teléfono móvil *Samsung Galaxy* S2 GT-I9100	600	2	0.3	24	15
Teléfono móvil *Huawei Ascend* G300	120	1	0.1	24	0.5
Teléfono móvil *Samsung Galaxy Ace* S5830	120	1	0.1	24	0.5
				Total	**154.66**

Tabla Apéndice I.III Costes asociados a la amortización de equipos informáticos

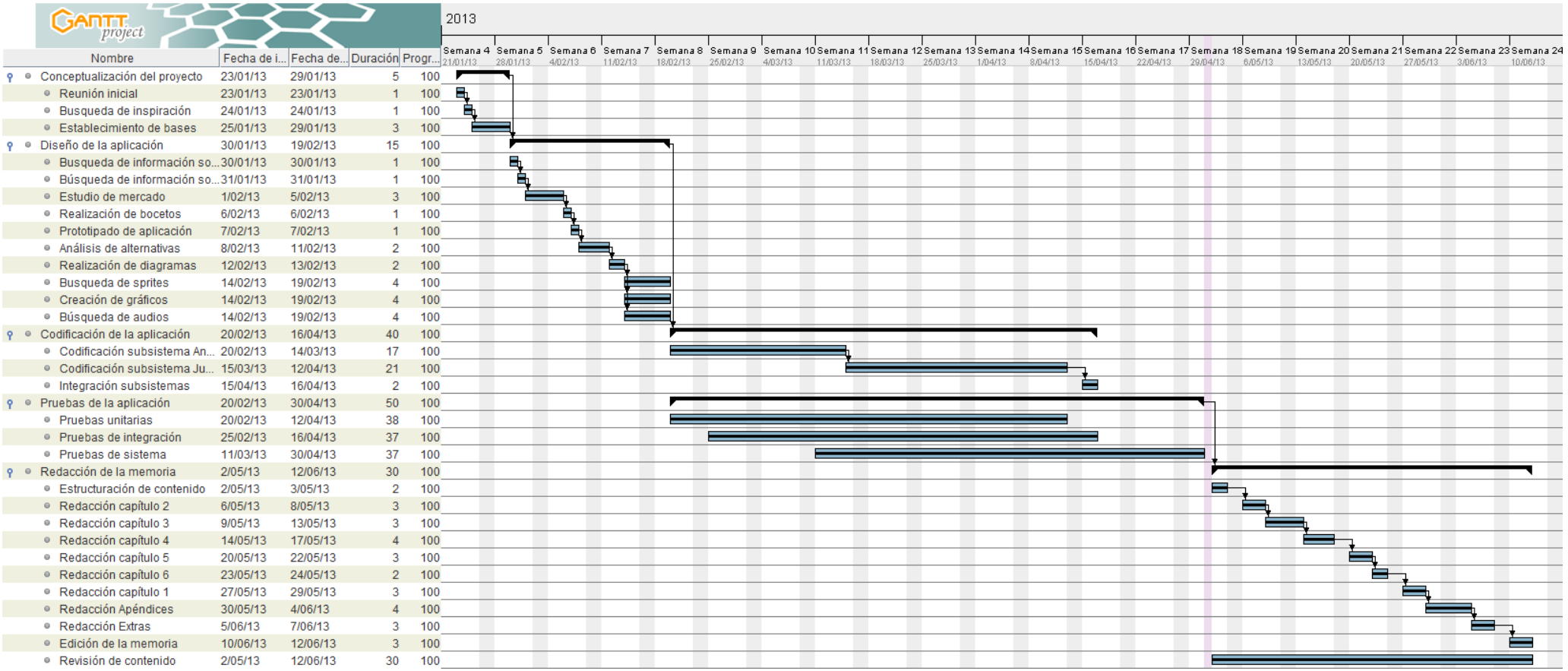
De este modo, sumando ambos costes obtenemos el presupuesto total de este Trabajo Fin de Grado y que es igual a 6489.66 Euros (Seis mil cuatrocientos ochenta y nueve Euros con sesenta y seis céntimos de Euro).

	Coste (€)
Costes asociados al esfuerzo humano	6335
Costes asociados a la amortización de equipos informáticos	154.66
Total	**6489.66**

Tabla Apéndice I.IV Costes totales asociados a la realización del proyecto

Apéndice II: Planificación del proyecto

A continuación se muestra el diagrama Gantt de la planificación del proyecto. Cabe destacar que a un día corresponden cuatro horas de trabajo.



Nombre	Fecha de i...	Fecha de...	Duración	Progr...
Conceptualización del proyecto	23/01/13	29/01/13	5	100
Reunión inicial	23/01/13	23/01/13	1	100
Busqueda de inspiración	24/01/13	24/01/13	1	100
Establecimiento de bases	25/01/13	29/01/13	3	100
Diseño de la aplicación	30/01/13	19/02/13	15	100
Busqueda de información so...	30/01/13	30/01/13	1	100
Búsqueda de información so...	31/01/13	31/01/13	1	100
Estudio de mercado	1/02/13	5/02/13	3	100
Realización de bocetos	6/02/13	6/02/13	1	100
Prototipado de aplicación	7/02/13	7/02/13	1	100
Análisis de alternativas	8/02/13	11/02/13	2	100
Realización de diagramas	12/02/13	13/02/13	2	100
Busqueda de sprites	14/02/13	19/02/13	4	100
Creación de gráficos	14/02/13	19/02/13	4	100
Búsqueda de audios	14/02/13	19/02/13	4	100
Codificación de la aplicación	20/02/13	16/04/13	40	100
Codificación subsistema An...	20/02/13	14/03/13	17	100
Codificación subsistema Ju...	15/03/13	12/04/13	21	100
Integración subsistemas	15/04/13	16/04/13	2	100
Pruebas de la aplicación	20/02/13	30/04/13	50	100
Pruebas unitarias	20/02/13	12/04/13	38	100
Pruebas de integración	25/02/13	16/04/13	37	100
Pruebas de sistema	11/03/13	30/04/13	37	100
Redacción de la memoria	2/05/13	12/06/13	30	100
Estructuración de contenido	2/05/13	3/05/13	2	100
Redacción capítulo 2	6/05/13	8/05/13	3	100
Redacción capítulo 3	9/05/13	13/05/13	3	100
Redacción capítulo 4	14/05/13	17/05/13	4	100
Redacción capítulo 5	20/05/13	22/05/13	3	100
Redacción capítulo 6	23/05/13	24/05/13	2	100
Redacción capítulo 1	27/05/13	29/05/13	3	100
Redacción Apéndices	30/05/13	4/06/13	4	100
Redacción Extras	5/06/13	7/06/13	3	100
Edición de la memoria	10/06/13	12/06/13	3	100
Revisión de contenido	2/05/13	12/06/13	30	100

Figura Apéndice II.I Diagrama de Gantt

Apéndice III: Tablas de pruebas de sistema

Las pruebas realizadas en el tramo de pruebas de sistema, tras la integración de ambos subsistemas, ya descritos en el cuerpo principal del documento, incluyen el seguimiento de unas pautas, detalladas a continuación, que recogen la información básica acerca de las mismas de forma estructurada y simple. Por cada prueba se adjunta una tabla indicando el resultado de la misma y que contiene los campos que ahora se detallan:

- ID: número identificador de la prueba. Es un número de tres dígitos incremental.
- Descripción: pequeña frase que indica el objetivo de la prueba.
- Pasos: pautas a seguir para realizar la prueba.
- Errores posibles: son los errores susceptibles de aparecer al realizar la prueba.
- Errores encontrados: son los errores encontrados tras la realización de la prueba.
- Estado: valoración acerca del resultado de la prueba. Puede ser "superada" o "no superada".

A continuación se muestra la plantilla de tabla que se utilizó para las pruebas.

ID	
Descripción	
Pasos	
Errores Posibles	
Errores Encontrados	
Estado	

Tabla Apéndice III.I Plantilla de pruebas de sistema

A continuación se mostrarán las tablas de las pruebas realizadas y sus resultados.

ID	001
Descripción	Inicio correcto de la aplicación
Pasos	1. Iniciar la aplicación
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.II Prueba de sistema 001

ID	002
Descripción	Inicio correcto del juego como Kirby
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Kirby
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.III Prueba de sistema 002

ID	003
Descripción	Inicio correcto del juego como Mario
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Mario
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.IV Prueba de sistema 003

ID	004
Descripción	Inicio correcto del juego como Pikachu
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Pikachu
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.V Prueba de sistema 004

ID	005
Descripción	Reinicio correcto de la partida como Kirby
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Kirby 4. Morir en el juego 5. Reiniciar la partida
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego. • Error al reiniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.VI Prueba de sistema 005

ID	006
Descripción	Reinicio correcto de la partida como Mario
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Mario 4. Morir en el juego 5. Reiniciar la partida
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego. • Error al reiniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.VII Prueba de sistema 006

ID	007
Descripción	Reinicio correcto de la partida como Pikachu
Pasos	1. Iniciar la aplicación 2. Pulsar botón de inicio 3. Seleccionar a Pikachu 4. Morir en el juego 5. Reiniciar la partida
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego. • Error al reiniciar el juego.
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.VIII Prueba de sistema 007

ID	008
Descripción	Capacidad de salto correcta
Pasos	1. Iniciar la aplicación

	2. Pulsar botón de inicio 3. Seleccionar un personaje 4. Pulsar la pantalla para saltar
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego. • Personaje no salta • Sensor de movimiento activado
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.IX Prueba de sistema 008

ID	009
Descripción	Apagado correcto de música
Pasos	1. Iniciar la aplicación 2. Pulsar botón de opciones 3. Activar la opción de apagado de música
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Música no se apaga
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.X Prueba de sistema 009

ID	010
Descripción	Apagado correcto de sonidos FX
Pasos	1. Iniciar la aplicación 2. Pulsar botón de opciones 3. Activar la opción de apagado de sonidos FX 4. Volver al menú principal 5. Pulsar botón de inicio 6. Seleccionar un personaje
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Sonidos FX no se apagan
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.XI Prueba de sistema 010

ID	011
Descripción	Cambio a modo música clásica
Pasos	1. Iniciar la aplicación 2. Pulsar botón de opciones 3. Activar la opción de cambio a música clásica
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Modo de música clásica no se inicia
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.XII Prueba de sistema 011

ID	012
Descripción	Apagado correcto de sonidos FX
Pasos	1. Iniciar la aplicación 2. Pulsar botón de opciones 3. Activar la opción de modo sensor de movimiento 4. Volver al menú principal 5. Pulsar botón de inicio 6. Seleccionar un personaje 7. Agitar el teléfono para saltar
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al iniciar el juego. • Personaje no salta • Sensor de movimiento desactivado
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.XIII Prueba de sistema 012

ID	013
Descripción	Finalización correcta de la aplicación
Pasos	1. Iniciar la aplicación 2. Cerrar la aplicación
Errores Posibles	• No se ha podido iniciar la música. • Error al crear el servicio. • Error al detener el servicio
Errores Encontrados	-
Estado	Superada

Tabla Apéndice III.XIV Prueba de sistema 013

Apéndice IV: Manual de usuario

Endless Tribute es un juego de género *endless run* para sistema operativo *Android*. El objetivo del juego es aguantar el máximo tiempo posible corriendo y esquivando enemigos para conseguir la puntuación más alta.

Introducción

Este manual de usuario le permitirá conocer todas las facetas de la aplicación Endless Run. Al final del mismo se incluyen unos pequeños consejos para alcanzar la mayor puntuación posible.

¿Cómo iniciar la aplicación?

Para iniciar la aplicación asegúrese de que se encuentra instalada en su dispositivo. Una vez se esté seguro acceder al menú del dispositivo y buscar la aplicación.

Figura Apéndice IV.I Menú principal Endless Tribute

Iniciar partida

Para iniciar una partida nueva, pulse en el botón inicio y, a continuación seleccione un personaje, entre los tres disponibles, pulsando sobre su imagen.

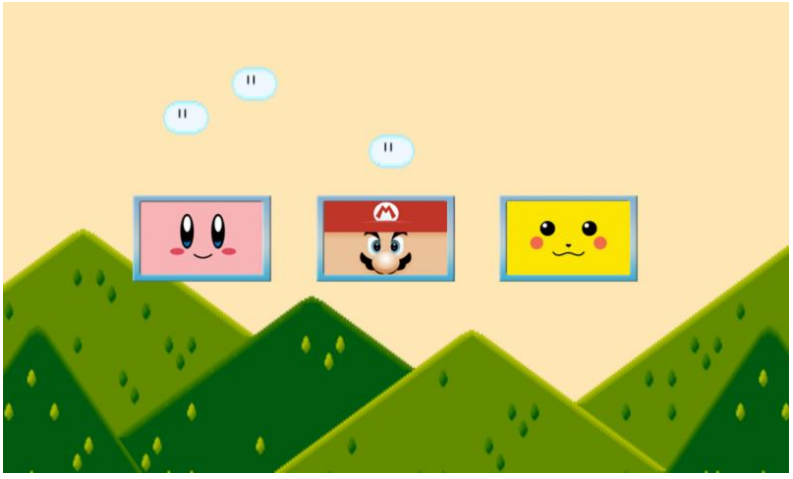


Figura Apéndice IV.II Menú de selección de personaje Endless Tribute

¿Cómo jugar?

Para jugar, únicamente es necesario hacer que el personaje realice un salto y de este modo esquive a los enemigos. Asimismo, en algún caso será necesario saltar para recoger las monedas. Si, por error, chocas con un enemigo la partida finalizará.

Modo de control normal

En este modo de control el salto se realizará tocando la pantalla táctil.

Modo de control por sensor

En este modo de control el salto se realizará agitando el dispositivo móvil hacia arriba.

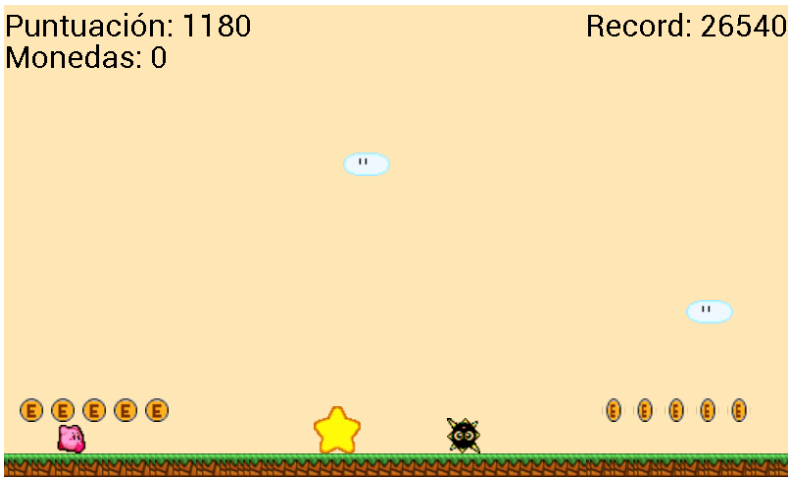


Figura Apéndice IV.III Pantalla de juego Endless Tribute

Reiniciar partida

Una vez hayas muerto, aparecerá una opción de reiniciar partida. Si pulsas el botón, la partida se reiniciará con el mismo personaje que hayas escogido previamente. Si lo que se quiere es iniciar partida con otro personaje, pulse el botón atrás del dispositivo y escoja el que desee.

Menús de opciones

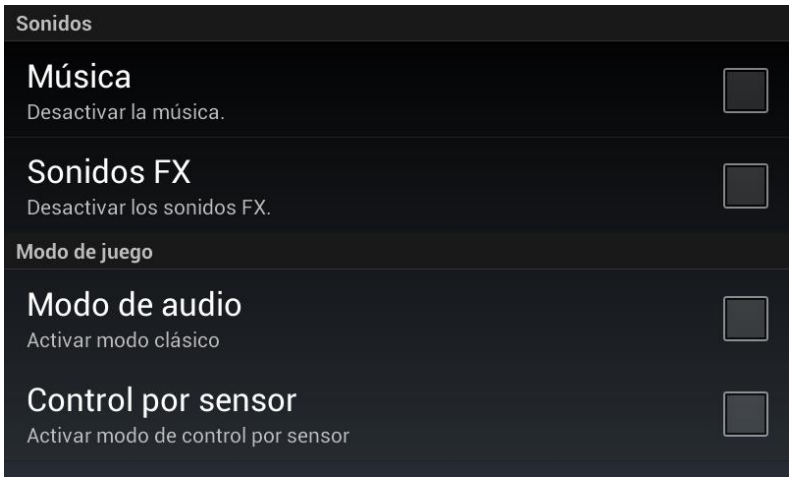


Figura Apéndice IV.IV Menú de opciones Endless Tribute

Entrar en el menú de opciones

Para entrar al menú de opciones, desde la pantalla principal, pulse en el botón opciones.

Opción *Música*

Si se activa esta opción, la música se apagará. Si, por el contrario, la opción se encuentra activada y se desactiva, la música volverá a sonar.

Opción *Sonidos FX*

Los sonidos FX son los diferentes "ruidos" que emiten tanto los personajes como el entorno. Si se activa esta opción, los sonidos FX se apagarán. Si, por el contrario, la opción se encuentra activada y se desactiva, los sonidos FX volverán a sonar.

Opción *Modo de audio*

Si se activa esta opción, se cambiará la música de toda la aplicación por la música clásica de *Ground Theme* de *Super Mario Bros*. Si, por el contrario, la opción se encuentra activada y se desactiva, la música volverá a ser la original.

Opción *Control por sensor*

Si se activa esta opción, se cambiará al modo de control por sensor. Si, por el contrario, la opción se encuentra activada y se desactiva, se volverá al modo de control normal.

Consejos

Para asegurar el salto, acérquese lo máximo posible al enemigo. Cuanto más se acerque más fácil le resultará saltarlo.

Los power up proporcionan un tiempo de invulnerabilidad y salto incrementado, hasta que se agota la mejora o bien hasta que te chocas con un enemigo.

Apéndice V: Encuesta de valoración de usuario

A continuación se adjunta el ejemplo de encuesta distribuido a los usuarios que se ofrecieron para probar el juego así como dos de los resultados de la misma.

Encuesta sobre satisfacción del usuario del videojuego Endless Tribute

Rellene esta encuesta con sinceridad. Muchas gracias.

Valore con un valor entre 1 y 5 siendo 1 completamente en desacuerdo y 5 completamente de acuerdo.

	1	2	3	4	5
Sobre la jugabilidad…					
¿Considera la dinámica del juego complicada?					
Valore la dificultad de Kirby (1 nada, 5 imposible)					
Valore la dificultad de Mario (1 nada, 5 imposible)					
Valore la dificultad de Pikachu (1 nada, 5 imposible)					
¿Considera el incremento de la velocidad excesivo?					
¿Considera que el juego responde bien a los toques en la pantalla?					
¿Considera que el juego responde bien a los movimientos del sensor?					
Sobre los menús…					
¿Considera que son sencillos e intuitivos?					
¿Entiende fácilmente lo que hacen las opciones del juego?					
¿Cumplen su función las opciones del juego?					

Valore las siguientes preguntas con si o no y, en caso afirmativo especifique.

	Sí	No
Sobre el juego…		
¿Le gustaría que apareciesen más personajes?	☐	☐
Si la respuesta es afirmativa indique quién/es:		
En caso de que el juego se comercializase, ¿pagaría por él?	☐	☐
Si la respuesta es afirmativa indique hasta cuánto:		
¿Le resultó útil el manual de usuario?	☐	☐

Figura Apéndice V.I Ejemplo de encuesta de valoración de usuario

Encuesta sobre satisfacción del usuario del videojuego Endless Tribute

Rellene esta encuesta con sinceridad. Muchas gracias.

Valore con un valor entre 1 y 5 siendo 1 completamente en desacuerdo y 5 completamente de acuerdo.

	1	2	3	4	5
Sobre la jugabilidad…					
¿Considera la dinámica del juego complicada?	x				
Valore la dificultad de Kirby (1 nada, 5 imposible)				x	
Valore la dificultad de Mario (1 nada, 5 imposible)	x				
Valore la dificultad de Pikachu (1 nada, 5 imposible)		x			
¿Considera el incremento de la velocidad excesivo?	x				
¿Considera que el juego responde bien a los toques en la pantalla?					x
¿Considera que el juego responde bien a los movimientos del sensor?				x	
Sobre los menús…					
¿Considera que son sencillos e intuitivos?					x
¿Entiende fácilmente lo que hacen las opciones del juego?				x	
¿Cumplen su función las opciones del juego?					x

Valore las siguientes preguntas con si o no y, en caso afirmativo especifique.

	Sí	No
Sobre el juego…		
¿Le gustaría que apareciesen más personajes?	x	
Si la respuesta es afirmativa indique quién/es:		
Peach		
En caso de que el juego se comercializase, ¿pagaría por él?	x	
Si la respuesta es afirmativa indique hasta cuánto:		
2 €		
¿Le resultó útil el manual de usuario?	x	

Figura Apéndice V.II Encuesta de valoración de usuario 1

Encuesta sobre satisfacción del usuario del videojuego Endless Tribute

Rellene esta encuesta con sinceridad. Muchas gracias.

Valore con un valor entre 1 y 5 siendo 1 completamente en desacuerdo y 5 completamente de acuerdo.

	1	2	3	4	5
Sobre la jugabilidad…					
¿Considera la dinámica del juego complicada?	x				
Valore la dificultad de Kirby (1 nada, 5 imposible)			x		
Valore la dificultad de Mario (1 nada, 5 imposible)		x			
Valore la dificultad de Pikachu (1 nada, 5 imposible)				x	
¿Considera el incremento de la velocidad excesivo?		x			
¿Considera que el juego responde bien a los toques en la pantalla?					x
¿Considera que el juego responde bien a los movimientos del sensor?			x		
Sobre los menús…					
¿Considera que son sencillos e intuitivos?					x
¿Entiende fácilmente lo que hacen las opciones del juego?					x
¿Cumplen su función las opciones del juego?					x

Valore las siguientes preguntas con si o no y, en caso afirmativo especifique.

	Si	No
Sobre el juego…		
¿Le gustaría que apareciesen más personajes?	x	
Si la respuesta es afirmativa indique quién/es:		
Sonic, Link, Samus, Snake		
En caso de que el juego se comercializase, ¿pagaría por él?	x	
Si la respuesta es afirmativa indique hasta cuánto:		
1 €		
¿Le resultó útil el manual de usuario?	x	

Figura Apéndice V.III Encuesta de valoración de usuario 2

Para consultar más resultados de la encuesta realizada, por favor, contacte al autor del documento mediante la siguiente dirección:

diego.trompeta@alumnos.uc3m.es